There are 2 parts to these notes, each addressing the topics of a year-long course in scientific computing. The courses are math475A and math475B at U. Arizona. Since the students taking this course sequence come from diverse backgrounds and most of them do not know any analysis, we have focused on scientific computing, rather than numerical analysis.

These notes were never developed for public consumption, and just like other sets of elementary scientific computing notes and textbooks, originality is not one of its strong characteristics. There are a few books which have strongly influenced what is presented here: the book by Atkinson on numerical analysis (I expressly use an old edition), Isaacson and Keller’s book which is available from Dover, my own student notes (I took numerical analysis from Ridgway Scott, Jinchao Xu, Douglas Arnold, and Todd Dupont, and attended classes by Stanley Osher while at UCLA).

An alternative set of notes that is worth looking at are those prepared by Cleve Moler, and are available freely from his Mathworks site.

The portion of these notes related to linear algebra is cursory. The reason is simple: there are people who can do a much better job at presenting this material and there are good books out there that cover that material. With regard to fundamentals I would strongly suggest Strang’s linear algebra book, as well as Trefethen and Bau’s book. Their geometric approach has revolutionized how applied linear algebra is taught. The geometric approach develops the type of insight fundamental to scientific computing.

These notes are being extensively revised. The major revisions are: (1) Incorporation of many more examples. (2) Restructuring of the course material. (3) Making greater use of hyperlinks in order to reduce the complexity of the notes, while at the same time making the cross references a useful feature of the text. (4) Changing the notes to PDF format.

As I said, these notes were never intended to be public. This is a work in progress, and as such, it is bound to have many errors, primarily of typographic nature (a regretful decision was to have these notes typed by someone
else who in turn faced the challenge of interpreting my own handwritten scribbles). I would be delighted to hear from you, if these notes have been of any use to you. Moreover, I am particularly interested in receiving corrections and suggestions.

At some point in time the use of matlab in numerical analysis classes was quite rare. I remember using a version of matlab back in 1987. It was still under development by Cleve Moler (and available only at Stanford). I started getting into the habit of writing code in matlab, debugging it, and only when it was debugged, porting the code to c/fortran/C++ (usually leaving all of the heavy lifting to fortran, the dynamic memory allocation to c, and the text/object oriented programming aspects to C++). It cut down the time I spent coding by a factor of 8-10, as compared to working directly with a "high level programming language."

I started using matlab for teaching in 1994 (while at UCLA) as did others. It was amusing to see some of my colleagues insist that matlab was just a toy and should not be used to teach, let alone compute; their silly claim was that real computing was done in c, C++, fortran, etc. My orthodox colleagues were able to have their better students write 4-5 high level codes during a typical semester; I was able to have most of my students write between 20-35, using matlab, focusing on scientific computing rather than programming.

Anyway, the use of matlab is no longer an issue (and I am glad that silly fight is over!). If you do not know matlab, you have little to worry, unless you are afraid to experiment. Matlab is an interpretive computer language and sometimes the best way to find out how something works is just to try it out\(^1\). Read the first 20 pages of the primer and do homework 1 of Math475A. This should take no more than 1:20 hours to do. This will teach you enough of the essentials of matlab to get going. The homeworks, beyond the first one, in math475A and B all start with a little primer on some aspect of matlab that will be useful.

The sequence followed in these notes results from two things: The assumption that matlab might be new to the student. Since we go from scalar, to vector,

\(^1\)You can consider OCTAVE, which is free. But if you are going to do this, you might as well consider PYTHON, which subsumes an interpretive scientific computing environment into a shell/macro environment. It is free and the web is replete with tutorials and lots of tools to use.
to matrix problems, the student will have time to learn enough matlab so that by the time matrix problems come around, they will be proficient. The more important reason, however, is that it is important to emphasize the notion of norm and of error. I found it better to work up in complexity, from scalar, to vector, to matrix norms.

Please note that the algorithms given in these notes are not matlab compatible with regard to indices. Be careful, if you are new to matlab, to adjust the algorithms for indices that include non-positive values.

I had help in the writing/editing of these notes by Prof. Rob Indik, Dr. Emily Lane, and Ms. Rachel Labes; their input and their hard work has yielded a better set of notes. Whatever errors are still in the notes are all my own. I do appreciate hearing from you, either with constructive criticism or to alert me of errors/omissions.

....Juan M. Restrepo

Contents

1 SOME USEFUL FACTS (PART 1) 8
    1.1 Continuity ................................................. 8
    1.2 The Intermediate Value Theorem ......................... 15
    1.3 Orders of Convergence .................................. 16
    1.4 Big $O$ and $o$ notation: ............................... 19
    1.5 Taylor Series ............................................. 25
       1.5.1 Power Series ......................................... 25
       1.5.2 Taylor Series ......................................... 25
       1.5.3 Analytic and Entire Functions ....................... 26
       1.5.4 Taylor’s Theorem ..................................... 27
## 2 COMPUTER ARITHMETIC

2.1 Rounding .................................................. 35
2.2 Machine Precision: ................................. 38
2.3 Propagation of Round-off Error in Arithmetic Operations ... 39
2.4 Some Types of Errors ............................... 41
   2.4.1 Rounding Error ................................. 41
   2.4.2 Truncation Errors ............................... 43

## 3 COMMENTS ON SCIENTIFIC COMPUTING

3.1 Code Design and Construction .................. 43
3.2 The Big Picture on Numerical Schemes ....... 46
   3.2.1 Reporting Errors: .......................... 49

## 4 SENSITIVITY AND CONDITIONING

4.0.1 Summary: Stability Consistency and Accuracy ...... 52
4.1 Backward Error Analysis .......................... 61

## 5 SOLUTION OF ALGEBRAIC EQUATIONS

5.1 Bisection Method ................................. 64
   5.1.1 The Basic Bisection Algorithm ............. 64
5.2 Regula-Falsi Method (False-Position Method) ........ 68
5.3 Newton Raphson Method (NR Method) ............. 71
5.4 Steffensen Method ................................. 76
5.5 The Secant Method ................................. 77
5.6 Fixed Point Iteration ............................. 79
5.6.1 Contraction Mapping ........................................... 95
5.7 Non-Simple Roots ................................................. 97
5.8 Zeros of Polynomials .............................................. 99
  5.8.1 Horner’s Method .............................................. 100
5.9 Müller’s Method and Complex Roots ......................... 104

6 NORMS OF FUNCTIONS 109

7 INTERPOLATION 113
  7.1 Preliminaries ................................................... 113
  7.2 Polynomial Interpolation ...................................... 115
  7.3 Chebyshev Polynomials ....................................... 125
  7.4 Newton’s Divided Differences and Interpolation .......... 129
    7.4.1 Interpolation at Equally-Spaced Points using Newton
           Divided Differences ......................................... 133
  7.5 Remarks on Polynomial Interpolation ....................... 136
  7.6 Spline Interpolation ........................................... 138
    7.6.1 Piecewise Linear Case ................................... 139
    7.6.2 Cubic Splines ............................................. 143
  7.7 Trigonometric Interpolation .................................. 149
    7.7.1 Fourier Series Review .................................... 149
    7.7.2 Interpolation using Fourier Polynomials .............. 156

8 NUMERICAL DIFFERENTIATION 160
  8.1 Divided Differences/Taylor Series ......................... 161
8.2 Derivatives via Lagrange Polynomial Interpolation . . . . . . . 163
8.2.1 The Effect of Round-off Error . . . . . . . . . . . . . . 168

9 NUMERICAL INTEGRATION 171
9.1 Quadrature Via Polynomial Approximation . . . . . . . . . 171
9.1.1 The Trapezoidal Rule: . . . . . . . . . . . . . . . . . . . 173
9.2 Simpson’s Rule: . . . . . . . . . . . . . . . . . . . . . . . . . 176
9.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 179
9.4 Gaussian Quadrature . . . . . . . . . . . . . . . . . . . . . . . 180
9.4.1 Using Legendre Polynomials on $[-1,1]$ . . . . . . . . . 181
9.5 Composite Numerical Integration . . . . . . . . . . . . . . . . 184
9.6 Some Adaptive Quadrature Methods . . . . . . . . . . . . . . . 188
9.7 Monte Carlo Method for Integrals . . . . . . . . . . . . . . . . 190

10 RICHARDSON EXTRAPOLATION 193

11 LINEAR ALGEBRA REVIEW 197
11.1 Vector Norms . . . . . . . . . . . . . . . . . . . . . . . . . . . 197
11.1.1 Matrix Norms . . . . . . . . . . . . . . . . . . . . . . . . 198
11.1.2 Matrix Norms (Optional, a bit more advanced) . . . . . 199
11.2 Eigenvalues and Eigenvectors . . . . . . . . . . . . . . . . . . 205
11.2.1 Matrix Norm and The Spectral Radius . . . . . . . . . . 207
11.3 The Condition Number and Error Estimates . . . . . . . . . . 209
11.4 Linear Systems (Continued) and Canonical Forms . . . . . . . 211
11.5 Eigenvalues and The Canonical Forms of Matrices . . . . . . . 214
11.5.1 Location of Eigenvalues: ........................................ 215
11.5.2 Bounds for Perturbed Eigenvalues ............................ 219
11.5.3 Eigenvalues of the Symmetric Tri-Diagonal Matrix ... 221

12 NUMERICAL LINEAR ALGEBRA ................................. 222

12.1 Direct Methods for Linear Systems: Gaussian Elimination . 223
12.1.1 Ill-conditioning and Finite Precision Errors ............... 229
12.1.2 Finite Precision: Pivoting and Scaling ...................... 230
12.1.3 The LU Factorization ......................................... 236
12.1.4 The Special LU’s: Doolittle, Crout’s, and Choleski Algorithms ... ........................................... 240
12.1.5 Permutation Matrix .......................................... 247
12.1.6 Other Special Matrix Types ................................. 251
12.1.7 Crout Factorization of a Tri-diagonal Linear System ... 254

12.2 Iterative Methods for Solving Algebraic Systems ............ 257
12.2.1 Newton’s Method for Nonlinear Systems .................. 259
12.2.2 Iterative Techniques for Solving Linear Systems ...... 260
12.2.3 The Conjugate Gradient Method ............................ 272
12.2.4 A Comparison of Convergence of Jacobi 12.2.2, Gauss-Seidel 12.2.2, SOR 12.2.2, and Conjugate Gradient 12.2.3 278

13 NUMERICAL EIGENVALUE PROBLEMS .......................... 279

13.1 The Power Method ................................................ 280
13.2 Inverse Power Method .......................................... 283
13.3 The Rayleigh-Ritz Method: ....................................... 284
13.4 The QR Method .................................................. 287
  13.4.1 The QR Factorization ..................................... 287
  13.4.2 Obtaining Eigenvalues from a QR Factorization ...... 291
13.5 Inverse Iteration ................................................. 292

14 DIFFERENCE EQUATIONS ........................................ 294
  14.1 Homogeneous Difference Equations ......................... 295
  14.2 Stable Difference Equations .................................. 299

1 SOME USEFUL FACTS (PART 1)

1.1 Continuity

A function \( f(x) \) is said to be continuous at a point \( c \) if

\[
\lim_{x \to c} f(x) = f(c).
\]

Loosely put this means that the value of the function at the point \( c \) is equal to what we would guess it should be when approaching \( c \) from either side.

A function is said to be continuous on a interval \([a, b]\) if it is continuous at every point on the interval \([a, b]\) and if the \( \lim_{x \to a^+} = f(a) \) and the \( \lim_{x \to b^-} = f(b) \). There are several different ways in which a function may fail to be continuous on an interval. Figures 1 and 2 show functions which are discontinuous on the interval \([0, 1]\). In all the cases the functions are not continuous at \( x = 0.5 \) but as we can see they are discontinuous in many different ways. The first function in figure 1 has a removable discontinuity. The function is

\[
f(x) = \frac{2x^2 - x}{2x - 1}.
\]

This is not defined at \( x = 0.5 \) but it is defined everywhere else as \( f(x) = x \).
Figure 1: Examples of (a) a removable discontinuity and (b) a jump discontinuity.
Figure 2: Two more types of discontinuities
The function just has a hole at \( x = 0.5 \). The second function in figure 1 is

\[
f(x) = \begin{cases} 
2x^2 & x \leq 0.5 \\
2 - x & x > 0.5 
\end{cases}
\]

This function has a jump discontinuity. The limit does not exist because the limit from the left is

\[
\lim_{x \to 0.5^-} = 0.5
\]

but the limit from the right is

\[
\lim_{x \to 0.5^+} = 1.5.
\]

Now, with reference to the figures, we see that \( f(x) \) jumps at this point. The first function in figure 2 is

\[
f(x) = \frac{1}{(x - 0.5)^2}.
\]

This function is discontinuous because as \( x \) approaches 0.5 the value of the function goes to infinity. The second function in Figure 2 is

\[
f(x) = \sin\left(\frac{1}{x - 0.5}\right).
\]

As \( x \to 0.5 \) this function oscillates more and more wildly between \( \pm 1 \) and so the limit as \( x \) approaches 0.5 is not defined. Even if we were to zoom in on the function near \( x = 0.5 \) we would never be able to say what the value of the function was at \( x = 0.5 \).

Now that we have discussed some of the ways in which functions can be discontinuous let us concentrate on continuous functions. If a function is continuous on the interval \([a, b]\) we say it comes from the class of functions \( C^0([a, b]) \). If a function and its first derivative are both continuous on the interval \([a, b]\) we say that \( f(x) \in C^1([a, b]) \). Likewise if the function and its first \( n \) derivatives are all continuous on the interval \([a, b]\) we say that \( f(x) \in C^n([a, b]) \).

Figure 3 gives an example of a function that is in \( C^0([0, 1]) \) but not in \( C^1([0, 1]) \). The function is

\[
f(x) = \begin{cases} 
2x^2 & x \leq 0.5 \\
\frac{x}{4} + 0.375 & x > 0.5 
\end{cases}
\]
Figure 3: A function that is $C([0,1])$ but not $C^1([0,1])$. We can see that although the function is continuous it is not 'smooth' because there is a sharp bend at $x = 0.5$. When we look at the derivative of the function we can see that it is not continuous because there is a jump in the derivative at $x = 0.5$. 
Figure 4: A $C^1([0,1])$ function but not a $C^2([0,1])$ function, $f(x) = (x - 0.5)^{1.3}$. Figure 5 shows the first two derivatives of $f(x)$.

From the picture of $f(x)$ we can see that the function itself is continuous on $[0,1]$ but there is a sharp bend at $x = 0.5$. When we look at the derivative of $f(x)$ there is a jump discontinuity at $x = 0.5$ so the first derivative is not continuous. Thus $f(x)$ is in $C^0([0,1])$ but not in $C^1([0,1])$.

Figure 4 shows an example of a function that is in both $C^0([0,1])$ and $C^1([0,1])$ but not in $C^2([0,1])$. The function is

$$f(x) = (x - 0.5)^{1.3}.$$  

We can see from Figure 4 that the function is continuous on $[0,1]$ which means that it belongs to the class $C^0([0,1])$. Figure 5 shows the first and second derivatives of $f(x)$. We can see that the first derivative is also continuous which means that $f(x) \in C^1([0,1])$. However the first derivative has a sharp point to it at $x = 0.5$ and the the second derivative is discontinuous at $x = 0.5$ so the function is not in $C^2([0,1])$.

If the function and all its derivatives are continuous on an interval $[a, b]$ we say that the function is in $C^\infty([a, b])$. An example of functions that are $C^\infty([a, b])$ are the polynomials. All polynomials are continuous on any bounded interval.
Figure 5: The first and second derivatives of $f(x) = (x - 0.5)^{1.3}$. The first derivative of $f(x)$ is continuous so $f(x) \in C^1([0, 1])$ but we can see a sharp point at $x = 0.5$. This becomes a discontinuity in the second derivative so $f(x)$ is not in $C^2([0, 1])$. 
The derivative of a polynomial is another polynomial and so also continuous on the same interval. Note that \( f(x) = 0 \) is a great function in this sense. Not only is it continuous but its derivative is the same as the function. So \( f(x) = 0 \) is very definitely in \( C^\infty([a,b]) \).

Another useful fact about functions that are in the class \( C^\infty([a,b]) \) is that they are also in \( C^n([a,b]) \) for all values \( n \). In fact there is an ordering of the sets

\[
C^\infty([a,b]) \subset C^m([a,b]) \subset C^{m-1}([a,b]) \subset \ldots \subset C^2([a,b]) \subset C^1([a,b]) \subset C^0([a,b]).
\]

1.2 The Intermediate Value Theorem

**Theorem**: (Intermediate Value Theorem) Suppose \( f(x) \in C[a,b] \), where \( k \) is a number between \( f(a) \) and \( f(b) \). Then \( \exists \) a number \( c \in [a,b] \) such that \( f(c) = k \)

Draw a figure to convince yourself of this fact.

**Rolle’s Theorem** Suppose \( f(x) \in C[a,b] \) and differentiable on \( (a,b) \). If \( f(a) = f(b) \) then \( \exists \) at least one \( c \in (a,b) \) such that \( f'(c) = 0 \).

Again, a figure will help illustrate this theorem.

**Theorem**: (Intermediate Value Theorem for Integrals) The simplest version of this theorem says. Let \( w(x) \) and \( q(x) \) be 2 functions, continuous on in interval \( x \in [a,b] \). Suppose also that \( q(x) \geq 0 \) for \( x \in [a,b] \). Then, there exists a number \( s \in [a,b] \) such that

\[
\int_a^b w(x)q(x)dx = w(s) \int_a^b q(x)dx.
\]

The proof follows from noting that if \( w_1 \leq w(x) \leq w_2 \) for \( x \in [a,b] \), and \( q(x) \geq 0 \) for \( x \in [a,b] \), the fact that

\[
w_1q(x) \leq w(x)q(x) \leq w_2q(x), \quad w_1M \leq \int_a^b q(x)w(x)dx \leq w_2M,
\]
where $M := \int_a^b q(x)dx$, and the intermediate value theorem for functions. That is,

$$w(s)M = \int_a^b w(x)q(x)dx.$$
Figure 6: Upper plot shows the iterates converging to $e^1$, as per (1). Lower plot confirms the estimate in (2).

This means that the difference between the $n+1$ iterate and the the $L$ in this case (which is $\exp(1)$) and the distance between the $n$ iterate and $L$ is comparable. We use this to estimate the "rate of convergence" and here we see it is linearly convergent. See Figure 6.

Example: Consider $x_{n+1} = x_n - \frac{x_n^2}{x_n^2 + x_{n-1}^2}$, $n = 2, 3, ...$

\[
x_0 = 20.0 \\
x_1 = 15.0 \\
x_2 = 14.64 \\
x_{33} = 0.54 \\
x_{34} = 0.27 \\
\text{etc.}
\]

This one has more rapid convergence than the previous case. In fact, numer-
ical experimentation will confirm that
\[
\frac{|x_{n+1}|}{|x_n|} \to 0.
\]
We say that the rate of convergence of this example is “super linear”.

Example: Consider
\[
x_{n+1} = \frac{1}{2}x_n + \frac{1}{x_n}.
\]
Start it with \( x_1 = 2 \). Then, by calculating,
\[
\begin{align*}
x_1 &= 2 \\
x_2 &= 1.5 \\
x_3 &= 1.416667 \\
x_4 &= 1.41421 \\
&\text{etc.}
\end{align*}
\]
The limiting value is \( L = \sqrt{2} \). Again, one can confirm the following numerically:
\[
\frac{|x_{n+1} - \sqrt{2}|}{|(x_n - \sqrt{2})^2|} \leq 0.36.
\]
This is a case of quadratic rate of convergence. We see that the distance between the \( n+1 \) iterate and the \( L \) is about as large as the distance between the \( n \) iterate and \( L \) to the second power. This ratio is a constant. See Figure 7.

Orders of Convergence

We use orders of convergence to indicate whether a computation converges and to determine how fast a computation converges. All things being equal, if we have two converging methods of computing \( L \), we would choose the faster one.

Linear
\[
|x_{n+1} - x^*| \leq K|x_n - x^*| \quad n \geq N \in \mathbb{Z}, \text{ here } c < 1
\]
Superlinear:
\[
|x_{n+1} - x^*| \leq \varepsilon_n|x_n - x^*| \quad n \geq N
\]
Figure 7: Second-order convergence plots for (3). The upper plot shows the iterates converging to the value $\sqrt{2}$ very quickly. The lower plot confirms the estimate in (4).

and $\varepsilon_n \to 0$.

**Quadratic:**

$$|x_{n+1} - x^*| \leq K|x_n - x^*|^2,$$

have $K$ not necessarily less than 1

**General case:** If

$$|x_{n+1} - x^*| \leq K|x_n - x^*|^\alpha$$

we say “at least $\alpha$-order convergent.”

1.4 Big $\mathcal{O}$ and $o$ notation:

Let $[x_n], [\alpha_n]$ be two sequences

Write $x_n = \mathcal{O}(\alpha_n)$,
if there exist $K$ and $n_0$ constants, such that $|x_n| \leq K|\alpha_n|$ when $n \geq n_0$.

Big $O$ gives an upper bound on $x_n$, to within a constant.

Example Figure 8 shows two sequences, as continuous functions of the parameter $n$. Their ratio at $n$ is tending toward a constant, provided $n$ is sufficiently large. While it is not true that these functions are, to within a constant, the same order, there is $n > n_0$ for which this is true:

$$\lim_{n \to \infty} \left| \frac{f(n)}{g(n)} \right| \leq K \text{ as } n \to \infty \quad g(n) \neq 0.$$ 

here.

On the other hand, we say that

$$f(n) = o(g(n))$$

if for each $\varepsilon > 0$ there exists an $n_0$ such that $0 \leq f(n) \leq \varepsilon g(n)$ when $n \geq n_0$.

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$
Figure 9: $f(n)$ becomes insignificant with respect to $g(n)$ as $n \to \infty$.

i.e. $f(n)$ becomes insignificant with respect to $g(n)$ as $n \to \infty$, see Figure 9.

Figure 9 illustrates two functions that relate to each other in order in the “little oh” sense.

Suppose $x_n \to 0$ and $\alpha_n \to 0$. If $x_n = \mathcal{O}(\alpha_n)$ then $x_n$ converges to 0 as rapidly as $\alpha_n$. If $x_n = o(\alpha_n)$ then $x_n$ converges to “more” than $\alpha_n$.

We also note that $f(x) = \mathcal{O}(g(x))$ as $x \to \infty$ means that there exists an $r$ and $K$ such that $|f(x)| \leq K|g(x)|$ whenever $x \geq r$. So, for example,

$$\sqrt{x^2 + 1} = \mathcal{O}(x) \quad x \to \infty$$

since $\sqrt{x^2 + 1} \leq 2x$ when $x \geq 1$

We note that $f(x) = o(g(x))$ $x \to x^*$ Thus

$$\lim_{x \to x^*} \left| \frac{f(x)}{g(x)} \right| = 0.$$ 

Exercise Show that: $\frac{n + 1}{n^2} = \mathcal{O}\left(\frac{1}{n}\right)$. 

21
Show that $e^{-n} = O\left(\frac{1}{n!}\right)$.

What can you say about the relative order between $\sin(x)$ and $x^5$, as $x \to 0$?

$$
\sin x = x - \frac{x^3}{6} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots
$$

$$
\sin x - \left(x - \frac{x^3}{6}\right) = O(x^5) \quad x \to 0
$$

same as $\left|\sin x - x + \frac{x^3}{6}\right| \leq C|x^5| \quad x \to 0$, $C$ constant.

\[\square\]

Example: Consider the following functions

\begin{align*}
    f(n) & = e^n \\
    g(n) & = e^{2n} \\
    h(n) & = 1 + 4e^{2n} \\
    k(n) & = 5 - e^n + n
\end{align*}

and show that $h(n) = O(g(n))$:

\[
\lim_{n \to \infty} \frac{h(n)}{g(n)} = \lim_{n \to \infty} \frac{1 + 4e^{2n}}{e^{2n}} = \lim_{n \to \infty} \frac{1}{e^{2n}} + 4 = 4
\]

\[\square\]

and that $k(n) = O(f(n))$:

\[
\lim_{n \to \infty} \frac{k(n)}{f(n)} = \lim_{n \to \infty} \frac{5 - e^n + n}{e^n}
\]
\[
\begin{align*}
\lim_{n \to \infty} \frac{5}{e^n} - 1 + \frac{n}{e^n} &= -1.
\end{align*}
\]

Consider \( h(n) \times k(n) \) and \( g(n) \times f(n) \):

\[
\begin{align*}
\lim_{n \to \infty} \frac{h(n) \times k(n)}{g(n) \times f(n)} &= \lim_{n \to \infty} \frac{(1 + 4e^{2n})(5 - e^n + n)}{e^{2n}e^n} \\
&= \lim_{n \to \infty} \frac{5 + 20e^{2n} - e^n - 4e^{3n} + n + 4ne^{2n}}{e^{3n}} \\
&= \lim_{n \to \infty} \frac{5}{e^{3n}} + \frac{20}{e^n} - \frac{1}{e^{2n}} - 4 + \frac{n}{e^{3n}} + \frac{4n}{e^n} \\
&= -4.
\end{align*}
\]

We know that \( h(n) \times k(n) = O(g(n)) \times O(f(n)) \) and we have just shown that it is also \( O(f(n) \times g(n)) \). This means that \( O(f(n))O(g(n)) = O(f(n)g(n)) \).

We can also see that \( f(n) = o(g(n)) \)

\[
\begin{align*}
\lim_{n \to \infty} \frac{f(n)}{g(n)} &= \lim_{n \to \infty} \frac{e^n}{e^{2n}} \\
&= \lim_{n \to \infty} \frac{1}{e^n} \\
&= 0.
\end{align*}
\]

If we consider \( f(n) + h(n) = e^n + 1 + 4e^{2n} \)

\[
\begin{align*}
\lim_{n \to \infty} \frac{f(n) + h(n)}{g(n)} &= \lim_{n \to \infty} \frac{e^n + 1 + 4e^{2n}}{e^{2n}} \\
&= \lim_{n \to \infty} \frac{1}{e^n} + \frac{1}{e^{2n}} + 4 \\
&= 4.
\end{align*}
\]

So \( f(n) + h(n) = o(g(n)) + O(g(n)) = O(g(n)) \).
So far we have only considered the limit as $n \to \infty$. We can also consider the limit as $\epsilon \to 0$. Consider
\[
\begin{align*}
    f(\epsilon) &= 1 + 4\epsilon^2 \\
    g(\epsilon) &= \sqrt{\epsilon} + 3\epsilon^3 \\
    h(\epsilon) &= \epsilon(1 - \sqrt{\epsilon})
\end{align*}
\]
We can see that $h(\epsilon) = o(g(\epsilon))$ (remember that we are now interested in $\epsilon \to 0$)
\[
\lim_{\epsilon \to 0} \frac{h(\epsilon)}{g(\epsilon)} = \lim_{\epsilon \to 0} \frac{\epsilon(1 - \sqrt{\epsilon})}{\sqrt{\epsilon} + 3\epsilon^3}
\]
\[
= \lim_{\epsilon \to 0} \frac{\epsilon(1 - \sqrt{\epsilon})}{\sqrt{\epsilon}(1 + 3\epsilon^2)}
\]
\[
= \frac{\sqrt{\epsilon}(1 - \sqrt{\epsilon})}{(1 + 3\epsilon^2)}
\]
\[
= 0
\]
We can also see that $g(\epsilon) = o(f(\epsilon))$
\[
\lim_{\epsilon \to 0} \frac{g(\epsilon)}{f(\epsilon)} = \lim_{\epsilon \to 0} \frac{\sqrt{\epsilon} + 3\epsilon^3}{1 + 4\epsilon^2}
\]
\[
= \lim_{\epsilon \to 0} \frac{\sqrt{\epsilon}(1 + 3\epsilon^2)}{1 + 4\epsilon^2}
\]
\[
= 0
\]
Now let us compare $h(\epsilon)$ and $f(\epsilon)$
\[
\lim_{\epsilon \to 0} \frac{h(\epsilon)}{f(\epsilon)} = \lim_{\epsilon \to 0} \frac{\epsilon(1 - \sqrt{\epsilon})}{1 + 4\epsilon^2}
\]
\[
= 0.
\]
So we can see that $h(\epsilon) = o(g(\epsilon)) = o(o(f(\epsilon))) = o(f(\epsilon))$.

Some rules for big O and little $o$ notation are as follows. We have already proven some of these rules in the examples above. See whether you can prove the remaining rules.

\[
O(o(f)) = o(O(f)) = o(o(f)) = o(f)
\]
\(O(fg) = O(f)O(g)\)
\(O(f) + o(f) = O(f)\)
\(O(O(f)) = O(f)\)
\(O(f) - O(f) = O(f)\)
\(O(f) + O(f) = O(f)\).

1.5 Taylor Series

1.5.1 Power Series

A power series, which is a special series, is an infinite series. A power series is of the form

\[ S(x; x_0) := \sum_{n=0}^{\infty} a_n(x - x_0)^n. \]

Here \(a_n\) and \(x_0\) are constants. A power series may or may not converge. The convergence depends on the form of the coefficients \(a_n\), but it also depends on where the center of the series \(x_0\) lies and how large the \(|x - x_0|\) interval is. There are a host of ways to test whether a series converges and over what interval. These tests are described in a good calculus book.

The custom is that if we are truncating the series or defining a finite-term series, that we specifically add the qualifier "finite" but otherwise we understand the series to be infinite. The finite series is of the form

\[ S_k(x; x_0) = \sum_{n=0}^{k} a_n(x - x_0)^n. \]

1.5.2 Taylor Series

A Taylor series is a special power series. It’s a power series expansion of a function \(f(x)\) about a point \(x_0\):

\[ f(x) = \sum_{n=0}^{\infty} a_n(x - x_0)^n = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 \ldots \]
\[
+ \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n + \ldots
\]

(if \( x_0 = 0 \) the series is called a Maclaurin series). Clearly,
\[
a_n = \frac{f^{(n)}(x_0)}{n!}, \quad n = 0, 1, 2, \ldots
\]

If a function depends on 2 variables \( x, y \), say, then
\[
f(x, y) = f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0)
+ \frac{1}{2!}[f_{xx}(x_0, y_0)(x - x_0)^2 + 2f_{xy}(x_0, y_0)(x - x_0)(y - y_0) + f_{yy}(x_0, y_0)(y - y_0)^2] + \ldots
\]

about \( x_0 \) and \( y_0 \). Here, \( f_x := \frac{\partial f}{\partial x} \), etc.

### 1.5.3 Analytic and Entire Functions

If \( f(x) \) is given by a convergent power series in an open disc centered at \( x_0 \), it is said to be **analytic** in this disc. For \( x \) in this disc, \( f \) is equal to a convergent power series:
\[
f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n.
\]

Differentiating by \( x \) the above formula \( n \) times, then setting \( x = x_0 \) yields the coefficients
\[
a_n = \frac{f^{(n)}(x_0)}{n!}.
\]

Hence the power series expansion agrees with the Taylor series. Thus a function is analytic in an open disc centered at \( x_0 \) if and only if its Taylor series converges to the value of the function at each point of the disc. If \( f(x) \) is equal to its Taylor series everywhere it is called **entire**.

Polynomial functions and the exponential function \( \exp(x) \) and the trigonometric functions sine and cosine are examples of entire functions.

Examples of functions that are not entire include the logarithm, the trigonometric function tangent, and the inverse tangent. For these functions the Taylor series do not converge if \( x \) is far from \( x_0 \). Taylor series can be used to calculate the value of an entire function in every point, if the value of the function, and of all of its derivatives, are known at a single point.
1.5.4 Taylor’s Theorem

The Taylor series of \( f(x) \) is a polynomial expansion of the function. Let \( k \geq 1 \) be an integer and let the function \( f: \mathbb{R} \to \mathbb{R} \) be \( k \) times differentiable at the point \( x_0 \in \mathbb{R} \). Then there exists a function \( h_k: \mathbb{R} \to \mathbb{R} \) such that

\[
f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2!}f''(x_0)(x-x_0)^2 + \frac{1}{3!}f'''(x_0)(x-x_0)^3 \ldots + \frac{1}{k!}f^{(k)}(x_0)(x-x_0)^k + h_k(x)(x-x_0)^k,
\]

with

\[
\lim_{x \to x_0} h_k(x) = 0.
\]

More compactly

\[
f(x) = p_k(x) + h_k(x),
\]

where \( p_k(x) \) is a polynomial of degree \( k \), and \( h_k(x) \) as above. If we have more regularity on \( f(x) \) we can estimate

\[
f(x) - p_k(x).
\]

We call this the remainder. Let \( f(x) \) be \( k + 1 \) times differentiable on the open interval and continuous on the closed interval between \( x_0 \) and \( x \), then

\[
R_k(x) := f(x) - p_k(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!}(x-x_0)^{k+1},
\]

for some \( \xi \) in between \( x \) and \( x_0 \).

2 COMPUTER ARITHMETIC

This section covers material that some years ago was extremely crucial. In the days of less powerful computers, the expense of accurate arithmetic calculations was high.

To appreciate the computational challenge we need to explain what we mean by \textit{accurate}: we mean that we can produce a number that is gets closer to the target number, possibly with a commensurate (linear) increase in the storage
or the speed of the resources (i.e. bigger machine), and/or more execution instructions.

Let’s discuss the challenges of performing even the most basic of arithmetic operations (plus, minus, times, divide) on a computing machine (a machine that can execute instructions and/or might have the ability to store information).

First, there’s the problem of just representing numbers: you could store these and read them off from the table every time you need them, or you can compute every number you need (i.e., execute instructions that yield the number, to a desired accuracy), or you can do a bit of both: store some, and compute others.

How many numbers need to be stored? ”An undefined many,” is an understatement. Also, how do we store an irrational number accurately? we could be clever here and say that we can store these as ratios of rational numbers. Or we could store numbers symbolically, for that matter. We then rely on programs or instructions to construct their representation and interpret how these symbols get modified by arithmetic operations and/or what other symbols they produce. We could even forgo storing anything. Just have a set of instructions that could generate the symbols as we need them and operate on them. It turns out that storing everything, even if it were possible, or following instructions, no matter how long it takes, are not realistic practical strategies.

We have developed a hybrid strategy to arithmetic computing. We create devices with finite storage capacity, capable of executing a finite but large enough set of instructions in a short time.

Some of these devices are discrete: for example, the abacus, the modern computer. Some of these are analog: the balance scale, an analog electronic calculator (made up of transistors, diodes, resistors, capacitors, and inductors).

There are 5 crucial aspects to machine computation:

1. what is the smallest/largest values stored or computed (the bandwidth),
2. what is the smallest discernible difference between close outcomes (the resolution),

3. how does this resolution change with the range of values being measured? (the sensitivity or condition number),

4. the speed (for example, measured in operations per unit time, throughput of information, etc).

5. the cost (for example, how much power the machine consumes, how many person/hours it requires to run or to maintain).

An ideal analog device can generate a range of continuous values (think of a mercury thermometer). The ideal discrete device is only capable of generating values from a predefined table of values (think of a digital clock). One could be tempted to say that an analog device might be better suited to handle arithmetic with the reals. However, there’s a practical issue: an outcome from the analog device needs to be sensed/interpreted, and thus the output will depend on the precision of the sensing/interpretation device.

A discrete device has a certain set of numbers (an ideal discrete device can have an infinite set of numbers, but there are finite jumps in value between neighboring numbers. Real digital devices have a finite set of numbers, spaced in equal or unequal intervals). Practical digital computers are not challenged by measuring or interpreting of outcomes as in the analog case. These discrete devices will have a fixed bandwidth, a resolution, and a sensitivity since there’s nothing from stopping us from using a variable grid to make choices on what numbers the device can represent. Virtually no discrete device is operated as a pure storage medium: it is also endowed with the capability of performing instructions and these can have an effect on the range of numbers it can represent.

In the days of simple machine calculators, small-word computer architectures, calculating with numbers was a tricky business since only a small and finite range of numbers could be represented on a small and finite-word machine. Moreover, the resolution of the number system was quite poor (i.e. what distinguished one number from a neighboring one as being different). Added to that was the problem of there being many ways of approximating numbers on the limited range. People devised very clever ways of calculating with acceptable precision, something that was especially important and motivated
by the war effort...before the second world war, the calculations concerned projectile trajectories, for example: structural calculations in the large engineering projects, such as the construction of Hoover Dam, and other public works, as well as aeronautical and maritime engineering design calculations. During the second world war there was a tremendous jump in the use of technology and thus in the need for large scale calculations. Check out the history of Los Alamos National Lab, for some historical details. These days the storage capabilities and speed of processing instructions has grown: the result is a bigger range of numbers and a finer resolution.

The goal of this section is to teach you the basics of how numbers are represented in machines; why approximation techniques need to be used; some surprising facts about machine number representation (such as the fact that they may not be evenly distributed along their range). We use these to understand a very fundamental issue to scientific computing: loss of precision...what is it, how do we identify it, how do we limit its effects, how does it affect what choice of machine we use to get the computation done, how it affects the design of computational algorithms.

Reals, integers, complex, are all represented with finite precision on practical computers/calculators.

**Floating Point Numbers:** refers to a standard representation of numbers on a computer. Take

\[
2346 = 2.346 \cdot 10^3 \\
0.0007236 = 7.236 \cdot 10^{-4}
\]

the decimal point moves or "floats" as the powers of the radix change.

We’ll use \( fp(x) \) to mean \( x \) in the floating point representation.

A floating point number representation system for numbers consists of 4 integers, plus a sign indicator:

- \( \beta \) base or radix
- \( t \) Precision
- \([L, U]\) exponent range

The sign indicator is a "bit" that indicates whether the number is positive or negative (yes, if the mantissa is zero, the sign indicator is ignored).
Base $\beta$ representation:

$$x = \pm \left( \frac{d_1}{\beta} + \frac{d_2}{\beta^2} \cdots \frac{d_t}{\beta^t} \right) \beta^e = \pm \underbrace{0.d_1d_2d_3 \cdots d_t \times \beta^e}_{\text{normalized representation}}$$

- $\beta$ base or radix
- $t$ Precision
- $[L, U]$ exponent range

where $0 \leq d_i \leq \beta - 1, \ i = 0, \ldots, t$

$L \leq e \leq U$

So then the string of base $\beta$ represented by $d_1d_2 \cdots d_t \equiv$ mantissa or significand and $e \equiv$ exponent or characteristic The \textit{normalized representation} refers to the fact that the first entry to the left on the mantissa is a 0. As we will see shortly, there are two equally acceptable choices for normalization when dealing with base 2 arithmetic, or “binary” arithmetic.

In a computer, the sign, exponent, the mantissa are stored in separate “fields,” where each field has a fixed width. These fields are measured in ”words,” which in turn have a prescribed number of bytes. Hence, a floating-point number system is finite and discrete.

Schematically, a number is stored in a computer, using one bit for the sign, $N$ bits for the exponent, and $M$ bits for the normalized mantissa. Hence, that a number will have a length of $1 + N + M$ bits on a particular machine. See Figure 10, for example.

Exercise: Show that the total number of machine numbers, for a machine operating in base $\beta$ with a mantissa of $t$ digits is $2(\beta - 1)\beta^{t-1}(U + L + 1) + 1$.

Binary Representation: here $\beta = 2$ thus $0 \leq d_i \leq 2 - 1$, or either 0 or 1.

Example $-8_{10} = -(2 \cdot 2 \cdot 2)_{10} = -1.00 \times 2^3$

Hexadecimal Representation: $\beta = 16$ $0 \leq d_i \leq 15$ where

$$d_i = 0, 1, 2, 3, \cdots, 9, A, B, C, D, E, F$$
Example

\[(23A.D)_{16} = 2 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 + 13 \times 16^{-1} = 512 + 48 + 10 + 0.8125 = 570.8125\]

Decimal Representation: \(\beta = 10\quad 0 \leq d_i \leq 9\)

Example: \(-(256)_{10} = -(2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0) = -256\)

Example: \(\left(\frac{1}{3}\right)_{10} = (0.3333 \cdots)_{10} = 0 \cdot 10^0 + 3 \cdot 10^{-1} + 3 \cdot 10^{-2} + 3 \cdot 10^{-3} \cdots \) □

Example: A reminder of basic conversion between a binary and decimal number.

\[(1101.0011)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = 8 + 4 + 0 + 1 + 0 + 0 + 0.125 + 0.0625 = 13.1875\]

The normalized representation of the number in binary and in decimal are, \((0.11010011)_2 \times 2^4\), and \(0.131875 \times 10^2\), respectively.

Example: Here is a reminder of how to convert from decimal to binary forms:

What is the binary form of \(x = \frac{2}{3}\)?

\[\frac{2}{3} = (0.a_1a_2a_3 \cdots)_2\]
\[\times 2\]
\[\frac{4}{3} = (a_1.a_2a_3 \cdots)_2 \Rightarrow a_2 = 1\], by taking the integer part of b.sides
\[\frac{4}{3} - 1 = \frac{1}{3} = (0.a_2a_3 \cdots)_2\]
\[\times 2\]
\[\frac{2}{3} = (a_2.a_3a_4 \cdots)_2\]
\[\times 2\]
\[
\frac{4}{3} = (a_2 a_3 a_4 \cdots)_2 \therefore a_2 = 0, a_3 = 1
\]
\[
\frac{4}{3} - 1 = \frac{1}{3} = (0.\overline{a_4 a_5 \cdots})_2
\]
\[
\times 2
\]
\[
\frac{2}{3} = (a_4 a_5 \cdots)_2
\]
\[
\times 2
\]
\[
\frac{4}{3} = (a_4 a_5 a_6 \cdots)_2 \therefore a_4 = 0 \quad a_5 = 1
\]
\[
\frac{2}{3} = (0.10101 \cdots) = (1.0101 \cdots) \cdot 2^{-1}
\]

How big are the fields used to store a floating point number? On every computer, it’s different. Here are some examples,

<table>
<thead>
<tr>
<th>System</th>
<th>(\beta)</th>
<th>(t)</th>
<th>(L)</th>
<th>(U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE HP, binary 16 bit word</td>
<td>2</td>
<td>11</td>
<td>-14</td>
<td>15</td>
</tr>
<tr>
<td>IEEE SP, binary 32 bit word</td>
<td>2</td>
<td>24</td>
<td>-126</td>
<td>127</td>
</tr>
<tr>
<td>IEEE DP, binary 64 bit word</td>
<td>2</td>
<td>53</td>
<td>-1022</td>
<td>1023</td>
</tr>
<tr>
<td>IEEE QP, binary 128 bit word</td>
<td>2</td>
<td>113</td>
<td>-16382</td>
<td>16383</td>
</tr>
<tr>
<td>Cray</td>
<td>2</td>
<td>48</td>
<td>-16383</td>
<td>16384</td>
</tr>
<tr>
<td>HP Calculator</td>
<td>10</td>
<td>12</td>
<td>-499</td>
<td>499</td>
</tr>
<tr>
<td>IBM 3000</td>
<td>16</td>
<td>6</td>
<td>-64</td>
<td>63</td>
</tr>
</tbody>
</table>

Here HP refers to ”half-precision”, SP refers to “single precision”, DP “double precision” and QP to ”quadruple precision.” The IEEE 754-1985 standard is the most commonly used standard, established in 1985, and it is used in just about every high level compiler and machine on the market. It can be adopted as a compiler option in high level language compilers (c, fortran, C++, etc). The first 4 are standard configurations, shared by a number of computers. The last 3 are products from large hardware companies which spearheaded high performance computing, programmable calculators, and multi-user main-frame computing. The point is that not everyone was using the same standards.

Note: a 32-bit machine achieves double precision calculation via software. DP calculations are usually slower than single precision calculations because these are done by software, rather than in hardware: more calculations and
Figure 10: Schematic representation of the storage on a machine that uses 32-bit word length. Here the mantissa is 24-bit in length, where the first bit = 1 (is implied, and we’re going to use the convention that the first bit is an implied 1). Each section is called a field -sometimes the sign bit is assigned to the exponent field-.

Data movement is involved. Intel produces 64-bit chips and these are presently embedded in just about any desktop or laptop.

Some important values in a floating point (fp) scheme:

- The smallest positive normalized fp number is called the “underflow level” (UFL) and it equals $\beta^L$.
- The largest positive normalized fp number is the “overflow level”, OFL and it is equal to $\beta^{U+1}(1 - \beta^{-t})$.

In binary, a floating point number is represented as

$$x = (-1)^s q \times 2^m,$$

here $s$ is the sign bit. Suppose we have a hypothetical computer (32-bit word length). Figure 10 illustrates schematically how a floating point number is stored. Computers use normalized binary representation (normalized, because this way the non-fraction part is always 0, or 1, therefore we do not to have to store it: most commonly the implied 1 is used).

We decide to normalize with an implied 1 to the left of the radix point. Hence, $q = (1.f)_2$, $s = 0 \rightarrow$ negative, or $s = 1 \rightarrow$ positive. $m = e - 127$ which is known as “8-bit biased exponent”. The range of $0 < e < (11 \ 111 \ 111)_2 = 2^8 - 1 = 255$; the number 0 is reserved for $\pm 0$, and 255 is reserved for $\pm \infty$,
and/or NaN, “not a number”. Since $m = e - 127$, then $-126 \leq m \leq 127$. The smallest positive number is $2^{-126} \approx 1.2 \times 10^{-38}$ and the largest positive number is $(2 - 2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$. The least significant bit is $2^{-23}$. Notice that the number set would not be evenly distributed on this machine. There are more numbers clustered around 0, since gaps of powers of 2 are smaller near 0 and larger closer to 2. And of course, the number of floating points available on this machine is finite!

Suppose we want to express $1/100$ on a particular machine. Is this number in the set available to the machine? (no) If not, what do we do?

First some definitions:

Suppose we are comparing two numbers $x$ and $\hat{x}$. Typically, we think of $x$ as the “exact” number and $\hat{x}$ as an approximation. We can then define two types of error:

Def: **Absolute Error** is the absolute difference between $x$ and $\hat{x}$, $|x - \hat{x}|$.

Def: **Relative Error** is the absolute error divided by the magnitude of $x$: $|x - \hat{x}|/|x|$.

### 2.1 Rounding

Def: **Machine Number** a number that is exactly representable on a particular machine (i.e., identical). If a number is not exactly representable as a floating point number on the machine, then it must be approximated by a nearby number.

The process of choosing a nearby number is called “rounding” and the resulting error is referred to as the “rounding error”.

Two common strategies:

- **Chopping**: Cut off $x$ at the $t$th digit: (a computationally cheap procedure).
• **Rounding:** rounding up/down a number, or in order to avoid bias, to set up a procedure such that the number in question is assigned the closest machine number.

**Example:** Here are two simple examples of chopping and rounding:

Take

\[ x = 475.846 \]  
approximation uses a 5 digit mantissa in base 10 arithmetic

\[ t = 5 \quad \beta = 10 \]

\[ x = 0.475846 \cdot 10^3 \] normalized

\[ fp(\hat{x}) = 0.47584 \cdot 10^3 \] chopping

\[ fp(\hat{x}) = 0.47585 \cdot 10^3 \] rounding.

Take

\[ x = 0.333333 \ldots \]

\[ fp(\hat{x}) = 0.33333 \] chopping

\[ fp(\hat{x}) = 0.33333 \] rounding.

**IEEE Standard Systems**

The IEEE standard supports several different rounding schemes, the most important and useful of which is: **Rounding-to-Nearest:** approximate to the closest \( fp \) number. If there’s a tie use \( fp \) number who’s last digit is even.

If the hardware doesn’t have this form of rounding “built in”, then the arithmetic must be done using software. This tends to slow things down to an impractical degree. Fortunately most processors now support IEEE and round to nearest rounding in hardware.

To see how rounding to the nearest works, consider the problem of representing the fraction \( 1/3 \). The binary representation is 0.101010\ldots Using an fp scheme this would fall between two machine numbers \( 0.10101010\ldots 10 \) and \( 0.101010\ldots 11 \). It is nearer to the latter, so it would be rounded up. In cases where the number to be rounded is precisely midway between two machine numbers, the IEEE round to nearest scheme rounds to the machine number whose mantissa’s least significant bit is a 0 (round to even on tie). Why is this done in this way? Why should this have ever have been a concern?
The problem this approach avoids is bias in the rounding error. If we follow the usual convention of always rounding up on a tie, there is a tendency to overestimate positive numbers (and underestimate negative ones) on average. You might feel that a tie is a very rare event, and thus can be ignored. This would be true if we were always working with arbitrary real numbers that needed to be rounded. However, in practice, most of the rounding is performed on numbers that are the result of arithmetic with other machine numbers. In these cases a tie is not very rare. (Why? Think about what happens when you add .10111011 to .10100000 using an 8 bit mantissa.)

In order to correctly round to the nearest machine number, at least two extra binary digits of information need to be generated, to distinguish between the four different cases: 1) the result is closer to the upper number, 2) the result is midway between, 3) the result is closer to the lower number 4) the result is equal to the lower number. It turns out that you need one more bit to deal with the possibility that the result of the operation is denormalized (the most significant bit is zeroed), and must be shifted before rounding.

As you can see, the issue of rounding is somewhat subtle, and quite a bit of effort has gone into devising good methods and hardware to deal with it. Details can be found in the IEEE standard.

There are libraries that can be gotten from netlib to correctly do rounding. Also libraries for “gradual underflow” exist. What are 2 nearby numbers on the hypothetical machine discussed earlier? Recall, we have a 24 bit mantissa - but 23 bits to worry about - with a 1 implied to the left of the radix point:

\[
\begin{align*}
    x_- &= (1.0101 \cdots 010) \cdot 2^{-1} \leftarrow \text{chopping} \\
    x_+ &= (1.0101 \cdots 011) \cdot 2^{-1} \leftarrow \text{rounding up}
\end{align*}
\]

\footnote{Gradual underflow preserves more mathematical identities. For example, \(x - y = 0\) implies \(x = y\) with gradual underflow. The former can hold without the latter with flush-to-zero. The performance impact is not obvious. Many "modern" processors do not accelerate arithmetic involving subnormal numbers, so gradual underflow seems slow. The additional identities give more room for programmers and compilers to optimize, however. A longer answer involves preserving relative error bounds in the face of underflow. Gradual underflow keeps simple codes robust.}
Which \( x \) do we take for \( \hat{x} \) of \( \frac{2}{3} \)?

Choose the closest:

\[
\begin{align*}
    x - x_- &= (0.1010 \cdots)_2 \cdot 2^{-24} = \frac{2}{3} \cdot 2^{-24} \\
    x_+ - x &= (x_+ - x_-) + (x_- - x) = \frac{1}{3} \cdot 2^{-24} \\
    \therefore \text{take } \hat{x} = x_+.
\end{align*}
\]

The absolute roundoff?

\[
|\hat{x} - x| = \frac{1}{3} \cdot 2^{-24}
\]

The relative roundoff

\[
\frac{|\hat{x} - x|}{|x|} = \frac{1}{3} \cdot \frac{2^{-24}}{2^{3}} = 2^{-25}
\]

### 2.2 Machine Precision:

Accuracy of a floating point system: use unit roundoff, machine precision, machine epsilon.

\[
\epsilon_{\text{mach}} = \beta^{1-t}
\]

for chopping

\[
\epsilon_{\text{mach}} = \frac{1}{2} \beta^{1-t}
\]

for rounding

The machine epsilon determines the maximum possible relative error in representing a number as floating point:

\[
\frac{|fp(x) - x|}{|x|} \leq \epsilon_{\text{mach}}
\]

Not to be confused with UFL: \( 0 < UFL < \epsilon_{\text{mach}} < OFL \), i.e. with underflow and overflow.

Example For IEEE single precision: \( \epsilon_{\text{mach}} = 2^{-24} \approx 10^{-7} \), “7 decimal digit precision”

double precision: \( \epsilon_{\text{mach}} = 2^{-53} \approx 10^{-16} \), sixteen decimal precision”

For chopping: let \( fp(x) = x (1 \pm \epsilon) \), \( \epsilon(x) \) in general
Assume (+):
\[
|x - fp(x)| = (0.0 \cdots 0d_t d_{t+1} d_{t+2} \cdots) \beta \times \beta^e
\]
\[
= (0 \cdot d_t d_{t+1} d_{t+2} \cdots) \beta \times \beta^{e-(t-1)} \leq \beta^{e-(t-1)}
\]
\[
\frac{(0.d_t d_{t+1} \cdots) \beta^e}{(0.d_1 \cdots) \beta^e} = \frac{(0.d_t d_{t+1} \cdots)}{(0.d_1 \cdots)} \beta^{-(t-1)} \leq \beta^{-(t-1)}
\]
\[
\frac{|x - x(1 + \varepsilon)|}{|x|} = \frac{|x\varepsilon|}{|x|} = |\varepsilon| \leq \beta^{-(t-1)}
\]

exercise: Work out the rounding case. □

Some examples: \( \varepsilon_{mach} \) for the IBM 3000: \( \varepsilon = 16^{-5} \). For the old CDC 6000: \( \varepsilon = 2^{-47} \)

Exceptional Values:
IEEE provides values for “exceptional” situations (floating point exceptions):

- "Infinite" Inf : divide \( n/0 \). There are 2 infinities, positive and negative. Note: in matlab: 1+Inf=Inf (it’s propagated sensibly)...this is not always the case on high level languages.

- "Not a Number" signified by NaN : \( 0/0, 0*Inf, Inf/Inf \) (Could be used by compiler to indicate that array is not defined). There are in fact 2 types of NaN’s. A quiet NaN and a signaling NaN. The latter of these carries a "bundle" which is useful in diagnostics/debugging.

There are "signed zeros" (+0 and −0, but these are not floating point exceptions).

2.3 Propagation of Round-off Error in Arithmetic Operations

Significant Digits:\(^3\) used often in place of relative error, to convey the accuracy of a number \( \hat{x} \), an approximation of the exact number \( x \).

\(^3\)A useful theorem for binary numbers:
We say that \( \hat{x} \) has \( m \) significant digits with respect to \( x \) if \( m \) is the largest nonnegative integer which

\[
\frac{|x - \hat{x}|}{|x|} \leq \frac{1}{2} \beta^{1-m}.
\]

In decimal notation this amounts to \( 5 \cdot 10^{-m} \).

Example Let \( x = 0.02136 \) \( \hat{x} = 0.02147 \) and assume we are using decimal notation: \( \beta = 10 \). Find the accuracy of \( \hat{x} \):

\[
\frac{|x - \hat{x}|}{|x|} = \frac{0.00011}{0.02136} \approx 0.00515 \leq 5 \cdot 10^{-2}
\]

Thus \( \hat{x} \) has 2 significant digits with respect to \( x \).

We’ve discussed the issue of representing numbers on a machine. We now bring up the following question: how is the accuracy of calculations affected by the use of approximate numbers?

Let the symbol * represent \( \{+,-,/,\cdot\} \), i.e. any of these operations, and let us assume that the calculations are being done on a finite precision floating point system. Let \( x \) be an exact real, \( \hat{x} \) a representation (approximation) of it. Let \( fp(\cdot) \) represent the operation whereby an exact number is converted into a floating point number on a finite-precision machine.

Then the following defines the "propagated" and the "round-off" error:

\[
(x \cdot y) - fp(\hat{x} \cdot \hat{y}) = (x \cdot y - \hat{x} \cdot \hat{y}) + (\hat{x} \cdot \hat{y} - fp(\hat{x} \cdot \hat{y}))
\]

\[
\text{propagated error} \quad \text{round off}
\]

Theorem (Loss of Precision of Binary Numbers): If \( x \) and \( y \) are positive normalized floating-point binary numbers such that \( x > y \) and

\[
2^{-q} \leq 1 - y/x \leq 2^{-p}
\]

then at most \( q \) and at least \( p \) significant binary bits are lost in the subtraction \( x - y \)
Example Consider the operation of a product of 2 numbers $x \times y$ using approximations of the numbers $x$ and $y$ to 3-digit rounding: the approximate quantities are $\hat{x} = 4.34$ and $\hat{y} = 1.02$, using 3-digit rounding.

The product of these, to 3-digit rounding: $fp(\hat{x} \times \hat{y}) = fp(4.34 \times 1.02) = fp(4.4268) = 4.43$. Thus, the "roundoff" error is 0.0032.

To obtain the "propagated" error:

$$|x - 4.34| \leq 0.005$$
$$|y - 1.02| \leq 0.005$$

this means $4.335 \leq x \leq 4.345$ $1.015 \leq y \leq 1.025$. Thus, $4.400025 \leq x \times y \leq 4.556125$. The propagated error is thus $-0.129325 \leq x \times y - \hat{x} \times \hat{y} \leq 0.026775$

\[\square\]

2.4 Some Types of Errors

2.4.1 Rounding Error

Just discussed. This type of error is the result of the inexact representation of numbers and the arithmetic operations upon them. This is usually due to finite precision and/or due to the approximation of all numbers on a machine in terms of a finite set of numbers (machine numbers). One of the most critical types of rounding errors are "loss of significance" errors.

Cancellation of significance, or loss of significance: This can occur when differencing nearby quantities, dividing very large by very small number, etc. The key thing is to recognize when this might happen in a calculation and figure out an equivalent way to do the calculation that is more stable, which in this case means without unduly loss of significance:

Example Suppose you want to find the roots of the quadratic polynomial equation:

$$ax^2 + bx + c = 0.$$ 

The roots are $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$.

There are several problems that can occur:
\( -b \pm \sqrt{b^2 - 4ac} \) could underflow.

- with regard to \( b^2 - 4ac \) (assume it is positive),
  - \( b^2 \approx 4ac \) could underflow.
  - \( a, b, c \) could be of very disparate size, thus could underflow/overflow.

The key is to manage \( b^2 \) and \( 4ac \) correctly, i.e. so differencing of nearby quantities is done carefully. Rescaling, the two roots,

\[
x_i = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{b^2 - (b^2 - 4ac)}{-2a(b + \sqrt{b^2 - 4ac})}
\]

\[
x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \quad x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}
\]

So, for \( b < 0 \), use

\[
x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}}.
\]

If \( b \geq 0 \), then

\[
x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{2c}{-b - \sqrt{b^2 - 4ac}}.
\]

So let’s see how this works, using 4-digit arithmetic with rounding. Take \( a = 0.05010 \), \( b = -98.78 \), \( c = 5.015 \). The roots, computing using 10-digit arithmetic are: 1971.605916 and 0.05077069387. Using 4-digit arithmetic \( b^2 - 4ac = 9757 - 1.005 = 9756 \), and \( \sqrt{b^2 - 4ac} = 98.77 \). If we use the standard formula:

\[
\frac{98.78 \pm 98.77}{0.1002} = \begin{cases} 1972 \\ 0.0998 \end{cases} 100 \% \text{ error due to cancellation.}
\]

Using the other formula

\[
\frac{10.03}{98.78 \pm 98.77} = \begin{cases} 1003 \leftrightarrow \text{large error} \\ 0.05077 \end{cases}
\]
So now that we know what can go wrong, we can code things in such a way so as to query the sign of \( b \) and choose the alternative formula that avoids the cancellations.

**Exercise** Consider the calculation of \( \exp(-\Delta H) \), where \( \Delta H := H_1 - H_2 \), and \( H_1 = \sum_{j=1}^{N} P_j^2 \), and \( H_2 = \sum_{j=1}^{N} Q_j^2 \). Here, \( N \) is a very large integer. Consider the four situations, when the set of \( \{ P_i \}_{i=1}^{N} \) and \( \{ Q_i \}_{i=1}^{N} \), are both less than 1, both much greater than 1, one much greater and one much smaller than 1, with respect to loss of significance. Devise algorithms that minimize loss of significance for each case.

**2.4.2 Truncation Errors**

These will be discussed in connection with every topic we will consider next. Examples of these errors are due to truncating series, replacing derivatives by differences, replacing function by a simpler polynomial, terminating iteration before convergence.

The truncation error *usually* dominates over the rounding error (is more prevalent or larger).

**3 COMMENTS ON SCIENTIFIC COMPUTING**

**3.1 Code Design and Construction**

First, some basic words about coding.

1) Decide on the solution scheme and make every attempt to make sure that it works. The most time consuming bugs to fix are mathematical, not computer bugs. The more work you do before actually coding something in order to assess whether it is going to work, the more time you’ll save yourself in the long run...some of the best computing is done AWAY from the computer!
Before you start coding up something in a high level language, such as C++, c, or Fortran, use matlab to try out the mathematics and the algorithm. Using matlab allows you to quickly debug and diagnose whether an algorithm works. In addition, it also forces you to experiment on small problems, which will speed up the debugging of the mathematical issues. Once you have the scheme mathematically debugged, you can easily translate the code to C, Fortran, C++, etc. Translating to a higher level language is not too painful when you can use the matlab code as a benchmark for the more complex code.

I consider all programming languages equally terrible and equally good: the key is to use the best features of all of them, which can be done. So, don’t be religious about programming languages, or afraid to try other languages. For example, I often program heavy duty computations in Fortran, but I get dynamic memory allocation by wrapping the Fortran in a c code. I also use c or C++ for all of my text handling and to create complicated things like classes and structures.

2) Often you are faced with the following question: should you develop a one-shot code or a production code?

A one-shot code is very typical in science and engineering. It is specifically designed to obtain results and then to be discarded. A production code is one that’s developed for repeated use or use by a number of other people.

In most cases a one-shot code is far less complex to write than a production code, because in the production code it becomes necessary to include instructions and features that make the code less prone to misuse by prospective users that might not understand or care to know what the code really does. The production code might have to be written in such a way so that it allows for backward compatibility, it might require extensible capabilities, and so on.

A one-shot code might be one that is used to obtain scientific results, usually as quickly as possible. There might be little point to developing a very fast code if it takes you years to do so; the scientific results are then years away. On the other hand, a production level code that is going to be run many times should be optimized for speed.

3) Make estimates of computing and storage requirements as early as pos-
4) Get used to writing pseudo-code to test ideas and keep you on the Big Picture.

5) Do not reinvent the wheel. There is plenty of good code, mostly developed by specialists in each particular algorithm or mathematical area. Of course, there’s also a dearth of bad stuff out there.

6) Keeping a modular design enables you to develop a library of software.

7) Date and put your name on each part of your code. Do not erase a piece of code until you know that the next version is always is superior, but keep them organized. There are a variety of version control packages (VCS).

8) A good rule of thumb is not to make any function or subroutine more than 1 page in length. This forces you to keep some modularity and makes program debugging a less messy affair.

9) Yes, comment thoroughly your code. However, I usually make the thorough comments ONLY after I know the code works well. Often times you put in comments that after the code has been worked on are no longer relevant...these comments just add confusion.

10) Use descriptive names for variables and don’t make them single letters...these are a pain to find using an editor when you are debugging.

11) Use complex coding only if it achieves an improvement on storage or speed. Gratuitous clever coding is useless.

12) Write README’s or MAN pages. These files are simple text files that contain basic usage instructions, changes, bugs, etc. These are useful not only to others but also to you. I often have trouble understanding my codes after I have not seen them in years.

13) Test, test, test your code.

14) If you are using a code to produce results for a research or term paper, embed in the paper a comment for each result/table/figure in which you remind yourself what code you used, where it is, what’s the input
file you used. This saves you a lot of aggravation if you have to return to this paper after a long while.

3.2 The Big Picture on Numerical Schemes

Most problems dealing with continuous variables (derivatives, integrals, etc) cannot be solved, even in principle, in a finite number of steps. So we need to find methods that find these approximately: Recast problem so that approximate problem (i.e. solved in a finite number of steps) is “close enough” or within some reasonable error tolerance. Sometimes it is impossible to reached a desired error tolerance due to the finite precision nature of machines.

In most cases we design a method of solution which is “convergent” to the exact solution. In order to assess whether a method is convergent we need to estimate its error and/or we need to define a sensible quantity which estimates its rate of convergence. Convergence may also depend on how the system is prepared.

Moreover, if a method is to be useful it should solve the problem efficiently, i.e. in the least amount of time and it must be reasonable with respect to computer resources.

Example Solving a very large linear algebraic system $Ax = b$, can be solved by direct methods, such as Gaussian elimination. As we shall see, however, if this problem is large enough, we might use an iterative method to solve for $x$. That is, a method that starts with an initial guess and by repeated iteration gives an approximate answer that is accurate enough. Why? it might be because we can obtain the answer in less time than by a direct method, and/or it might be that the direct method is prone to numerical round-off errors.

In order to make this judgement we will first learn to estimate how many operations are required to obtain a solution via a direct method, and what types of errors might occur. Invariably, the structure of $A$ and $b$ are also important.

What is the goal in the design of good numerical schemes? Solving problems...
EXACTLY is sometimes not possible and/or not even sensible (e.g. a chaotic system). What you want to do is to produce a numerical approximation which will have the property that the more you push it, the “better” the answer gets...i.e. the closer it gets (in some norm) to the solution.

The most important qualities of a good numerical schemes are:

- **Convergence:** you want the answer to approach the true solution as the algorithm or calculation is refined.
- **Stability:** you want your algorithm or calculation to be robust. That is to say, small changes in the input (or parameters) should not lead to large changes in the output.
- **Efficiency:** roughly speaking, there are two resources you’ll use: time (as measured in compute time) and storage (memory). You want to get an answer with reasonable usage of these two resources.
- **Consistency:** if the approximation and truncation error parameters need to be optimized in a certain way in order to obtain a certain result, your algorithm is fragile and possibly consistent ONLY under certain circumstances. Avoid this.

We will discuss these throughout the year in different contexts. Same with stability, but it is appropriate to mention this detail at this point:

**Stability:**

Suppose you are monitoring the error of a calculation $E_n$, as a function of the number of steps $n$ taken in a calculation. Let $\varepsilon$ be the initial error. If

$$ |E_n(\varepsilon)| \approx C n \varepsilon $$

then the error grows at most linearly, with $n$. We consider this an error growth that is acceptable. On the other hand, if

$$ |E_n(\varepsilon)| \approx k^n \varepsilon \quad \text{and} \quad k > 1, $$

or

$$ |E_n(\varepsilon)| \approx \varepsilon n^p, \quad p > 1, $$

47
Figure 11: *Linear growth of the error, with the number of steps, is acceptable.*

we have an error growth that is unacceptable. We call this situation "unstable." See Figure 11.

**General Strategy**

As we said, we will replace the problem with an approximate one with one with closely related solutions. For example, replace integrals with finite sums, replace derivatives with finite differences, replace matrices with simpler matrices; complicated functions with sequences of polynomial, differential equations with difference algebraic equations, nonlinear problems with linearized approximations, high-order differential equations with low order difference equations, and invariably on finite precision machines, infinite-dimensional spaces with finite-dimensional spaces.

This replacement must be done while preserving the solution in some sense. What should be preserved? It depends on the problem, as we shall learn.

**Sources of Approximations in Scientific Computation**

**Before Computing:**

Modeling (equations capturing some aspect of the physical world.)
Empirical Measurement (Lab data): finite precision, small sample size, random noise, etc.
Fundamental scientific theory and mathematical theorems.

At the computing stage

Error Analysis

\[
\begin{align*}
&\text{Approximation: replace a complicated process by a simpler process} \\
&\text{Truncation or discretization.} \\
&\text{Rounding: computer representation of real numbers and arithmetic operators.}
\end{align*}
\]

The aim in practical numerical analysis is seldom to compute exactly. Instead, it is about converging computing that is stable, efficient, and for which errors are estimated as exactly as possible.

3.2.1 Reporting Errors:

With practice we will be able to report either error in terms of either absolute or relative terms. Depending on the context presented by the problem we are trying to solve, it might make more sense to report the error in terms of just distances, and in some others, in terms of normalized distance. Hence, we have

Absolute and Relative Error

Abs error = approximate value – true value (could use absolute value)

\[
\text{Rel error} = \frac{\text{absolute error}}{\text{true value}}
\]

You must always define what these are, as the above definitions are not standardized.

A situation that illustrates how a dilemma arises in reporting errors is the following: suppose you have an estimator of the number of bacteria in a petri dish. If the estimate is that the error, defined as the absolute difference between the estimator device and the true population $10^3$, does this mean that the estimator is an inaccurate device? Well, in this context, it is probably more sensible to report a relative error, normalizing the absolute error by the typical populations that this device will normally encounter. An
estimate of this typical population could be the actual population. Suppose that this population was in the $10^6$ range. In that case the relative error is $10^{-3}$. Which error would make more sense to use as a qualifier to the accuracy of the estimator? It is up to you.

\[ \square \]

4 SENSITIVITY AND CONDITIONING

The stability of a computation affects the accuracy of the outcomes because it introduces a certain degree of uncertainty. Roughly speaking, inaccuracies in the input or in the methods we use to compute lead to inaccuracies in the output. A stable calculation is one in which uncertainties grow at most linearly, rather than exponentially, with the number of operations, or the sources of imprecision. If they grow exponentially, we say that the computation is ”sensitive,” and we might convey this by computing its ”conditioning.”

Identifying/computing the condition number of the algorithm is calculation, not only because it tells us something about the calculation, but because it is often the case that there might be several ways to compute the same thing and one should choose the most stable alternative, keeping in mind other constraints (for example, the size of the machine, the cost of the computation, etc).

What do we mean by computation? we mean the choice of arithmetic operations, the precision in which we choose to work in, the algorithm, the formula of the computation itself, the software implementation. We are concerned with the stability of calculations. These are of concern because we are generally computing approximately:

- because we are using approximations to numbers (we already discussed this), due to finite-precision/finite-memory.
- we might be using approximating formulas (for example, we are using a truncated Taylor series),
- we might be taking a finite sequence as an approximation to a full (perhaps infinite) sequence.
• we are using an approximate algorithm (we require efficiency as well, so we might settle for a an algorithm that gives approximate answers but is faster than an alternative).

• we are using a particular piece of software.

Hence, when we compute the sensitivity of a computation, we describe what the computation is sensitive to: for example, a computation may be sensitive to finite precision, it might be sensitive to the number of iterations you perform to reach a result, it might be sensitive to the dimension of the problem you are computing. It might also be sensitive to the choice of algorithm you opt for in accomplishing the computation.

We need to distinguish between the stability of a mathematical "formula" or a physical law, or desired outcome, and the stability of its computation (we are focusing on the computation: the algorithm and its computing machinery). We usually report the total sensitivity of a computation, as the sensitivity to the manner in which it is computed plus the sensitivity of the formula itself (to inputs, to parameters, etc).

Consider the following: suppose your company is asked to build a structure that moves motor vehicles across a road, in such a way as to not disrupt the other road’s traffic. The desired outcome is to move cars and trucks from one side to the other side of the road. There are several ways to do this, one might be a bridge, another a tunnel. Given your expertise and your cost estimate, you choose to deliver a bridge: this is already a sensitivity calculation outcome: the method called "tunnel" is more expensive (let’s pretend it is) and it is sensitive to your knowledge base than the "bridge" method, meaning that the outcome may be less accurate. In this case accuracy is defined as making it possible for any motor vehicle to make it from one side of the road to the other (regardless of the expense or the challenge in doing so). So the method we will use is the bridge. But there are many ways to build this bridge and many different materials that can be use in building it, (i.e., there are many ways to "compute" or achieve this). In this case the sensitivity should be measured with respect to the traffic on the bridge. We don’t want a sensitive bridge, meaning that the perturbations due to the passage of a little moped should not lead to exponentially large deflections of the bridge. However, we might be content with the bridge deflecting in a commensurate way to the amount of traffic (so long as it is reasonably
accurate, i.e. most motor vehicles will make it over the bridge): if increases in the input force lead to linear increases in the bridge vibration, we say that the bridge is well-conditioned or stable to traffic density perturbations. Sure, a bridge that does not deflect at all due to traffic would be terrific, but impractical or impossible; but more importantly, not worth considering, if the vibrating alternative is reasonably accurate (sure, you can always find a load that will break the bridge). The upshot is that the stability of this bridge will depend on the choice of design (hanging, girded, cantilevered, etc), and the choice of materials you put into it.

How do we quantify sensitivity? via a condition number, for example. It turns out that it is up to you to propose a sensible notion of sensitivity for each problem you work on, mostly guided by common sense and context. The condition number is a single number that conveys sensitivity, but you might imagine that a single number cannot completely convey the complexity of the sources of instability. Moreover, the condition number does not discern between instabilities due to the computation and the desired outcome, unless these two can be approximated by a linear relation.

4.0.1 Summary: Stability Consistency and Accuracy

Accuracy: a measure of the closeness of computed to the true solution.

\[ \left\{ \begin{array}{l}
\text{Stability: used to categorize problems and algorithms} \\
\text{Conditioning: categorizes algorithms and methods}
\end{array} \right. \]

An algorithm is stable if small perturbations to the input lead to small changes in the output.

An algorithm should also be consistent: the algorithm might not be computing what you think it is. On simple problems this is not hard to verify, but it is not trivial to verify in complex cases.

\[ \text{Stable and Consistent } \Rightarrow \text{ Convergent} \]

If the algorithm is stable, the output is exact for a nearby problem but the solution to a nearby problem is not necessarily close to the solution to
the original problem UNLESS problem is stable. Therefore, inaccuracy can result from applying a stable algorithm to an ill-conditioned problem as well as from applying an unstable algorithm to a well-conditioned problem.

If an algorithm is convergent, then the algorithm can achieve arbitrarily close outputs to the true solution, by increasing the resolution, storage, and the precision of the computation.

So fundamental to scientific computing is to estimate how accurate a computation is. Doing so requires us to show that our methods are stable and consistent, and that the more computing time and higher resolution leads to increasingly accurate results.

In what follows we will consider simple notions of sensitivity (linearized sensitivity). A natural way to qualitatively determine the effect of an individual error on the final result might be through derivatives. Here we are going to compute the sensitivity of the mathematical formula \( f(x) \):

Compute \( f(x) \) (for simplicity a \( C^2 \) function) and use \( \hat{x} \) (assume \( \hat{x} \) close to \( x \) and within the range of values for which \( f(x) \) has the above continuity)

\[
f(x) = f(\hat{x}) + \frac{df(\xi)}{dx}(x - \hat{x}) \quad \text{by the mean-value-theorem}
\]

\( x \leq \xi \leq \hat{x} \)

\( \text{Abs } (f(\hat{x})) = |f(x) - f(\hat{x})| = f'(\xi)|x - \hat{x}| \approx f'(\hat{x}) \cdot \text{Abs } (\hat{x}) \)

**Example** \( f(x) = \sqrt{x} \therefore f'(x) = \frac{1}{2\sqrt{x}} \therefore \)

\( \text{Abs } (\sqrt{\hat{x}}) \approx \frac{\text{Abs}(\hat{x})}{2\sqrt{\hat{x}}} \). For relative error:

\( \text{Rel } (\sqrt{\hat{x}}) = \left| \frac{\text{Abs}(\hat{x})^{1/2}}{\sqrt{\hat{x}}} \right| \approx \frac{\text{Abs}(\hat{x})}{2\sqrt{\hat{x}}} \cdot \frac{1}{\sqrt{\hat{x}}} = \frac{1}{2} \text{ Rel } (\hat{x}) \)

Hence, since the relative error in \( \sqrt{x} \approx \frac{1}{2} \) relative error in \( \hat{x} \) thus \( \sqrt{\hat{x}} \) is a safe operation.

\( \square \)
Suppose result depends on more than one variable, say \( x, y \)?

\[
f(x, y) = f(\hat{x}, \hat{y}) + \frac{\partial f}{\partial x}(\hat{x}, \hat{y})(x - \hat{x}) + \frac{\partial f}{\partial y}(\hat{x}, \hat{y})(y - \hat{y})
\]

\[
\therefore \text{Abs}(f(\hat{x}, \hat{y})) \approx |\frac{\partial f}{\partial x}(\hat{x}, \hat{y})| \text{Abs}(\hat{x}) + |\frac{\partial f}{\partial y}(\hat{x}, \hat{y})| \text{Abs}(\hat{y})
\]

Example Suppose you are building capacitors out of series of capacitors. You know that the capacitance \( z \) of a series of capacitors \( x \) and \( y \) is

\[
z = \frac{xy}{x + y}
\]

Suppose you have the following capacitors:

\( \hat{x} = 2.71\mu F \quad \hat{y} = 3.14\mu F \quad \) to 3 digits \( \therefore \) \( \text{Abs}(\hat{x}) = \text{Abs}(\hat{y}) = 0.005\mu F \)

computed \( \hat{z} = \frac{\hat{x}\hat{y}}{\hat{x} + \hat{y}} \)

\[
z = f(x, y) \quad \Rightarrow \quad \frac{\partial f}{\partial x} = \frac{y^2}{(x + y)^2} \quad \frac{\partial f}{\partial y} = \frac{x^2}{(x + y)^2}
\]

\[
f_x(\hat{x}, \hat{y}) \approx 0.288103 \quad f_y(\hat{x}, \hat{y}) \approx 0.214599
\]

\[
\therefore \text{Abs}(f(\hat{x}, \hat{y})) \approx 0.288103 \cdot \text{Abs}(\hat{x}) + 0.214599 \cdot \text{Abs}(\hat{y}) \approx 0.00251351
\]

Some problems are difficult to solve accurately not because an algorithm is ill-conceived but because of some inherent property of the problem. Even with exact arithmetic, the solution may be highly sensitive to perturbations.

A problem is insensitive or well-conditioned if the relative change in the input causes a reasonably commensurate change in the output.

Otherwise the problem is sensitive or ill-conditioned

One qualitative way to assess sensitivity is by computing a “condition number”. The condition number \( \equiv \text{Cond} \)

\[
\text{Cond} = \frac{|\text{relative change in solution}|}{|\text{relative change in input}|} = \frac{|[f(\hat{x}) - f(x)]/f(x)|}{|((\hat{x} - x)/x)|}
\]

54
where we’re assuming that $\hat{x} \approx x$.

Therefore,

if $\text{Cond} \gg 1$ then algorithm or function is ill-conditioned. On the other hand, if $\text{Cond} \ll 1$ then we say that the algorithm or function is well-conditioned, i.e., the relative output perturbation is less sensitive to the relative input perturbation.

\[ \text{Cond} (x) = \left| \frac{hf'(x)/f(x)}{h/x} \right| = \left| \frac{xf'(x)}{f(x)} \right| \]

Suppose $f(x) = e^x$

<table>
<thead>
<tr>
<th>Absolute error</th>
<th>$e_a = e^{x+h} - e^x \approx he^x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative error</td>
<td>$e_r = \frac{e^{x+h} - e^x}{e^x} \approx \frac{h e^x}{e^x} = h$</td>
</tr>
<tr>
<td>Cond($x$)</td>
<td>$</td>
</tr>
</tbody>
</table>

**Example:** Here’s an example of a very sensitive operation. Computing the cosine of a number! You might wonder what’s so sensitive about that? well, this answer illustrates the fact that not only the operation is important, but also the range of values $x$ of the input.

Compute $\cos x$ where $x \approx \pi/2$

$\hat{x} = x + h$

$e_a = \cos(x + h) - \cos(x) \simeq -h \sin(x) \approx -h$
\[ e_r = -h \tan(x) \approx \infty \]

therefore small changes in \( x \) near \( \frac{\pi}{2} \) cause large relative changes in \( \cos x \) regardless of the method used in the computation.

\[
\begin{align*}
\cos(1.57079) &= 0.63267949 \cdot 10^{-5} \\
\cos(1.57078) &= 1.63267949 \cdot 10^{-5}.
\end{align*}
\]

Examining the relative change in the input and the output:

\[
\begin{align*}
\text{ratio of inputs} &= 1.00000636 \\
\text{ratio of outputs} &= 2.5805791
\end{align*}
\]

\[ \square \]

Let us think of the problem as an engineer would: we have an input \( x \), which is a vector of size \( m \) and an output \( y = f(x) \) which is a vector of size \( n \). The black box is \( f \). We would like to know if we make small changes on the input \( x \), what kind of changes do we obtain in the output \( y \)? Are they large changes, or are they proportional to the size of the perturbation? (say, if the input plus perturbation was \( x + \delta x \), where \( \delta x \) is also an \( m \) dimensional vector of small size, is the perturbation in \( y \), call it \( \delta y \) which is of size \( n \), is \( \delta y \) to within a multiplicative constant the same as \( \delta x \)? If so, we say the problem is very well conditioned...how much ill-conditioning we tolerate is a matter of experience and context.

Take the case when the have \( y = f(x) \), where \( x \) and \( y \) are scalars. We assume that \( f'(x) \) exists, and we write, by Taylor’s theorem,

\[
\delta y = f(x + \delta x) - f(x) = f'(\xi) \delta x,
\]

where \( \xi \) is some value between \( x \) and \( x + \delta x \). We then form

\[
\frac{\delta y}{y} \approx \frac{xf'(x) \delta x}{f(x) \cdot x}.
\]

The approximation happened in where we evaluate the derivative...we are assuming that \( \delta x \) is small but finite, by the way. The relative error is just
the absolute value of this:
\[
\left| \frac{\delta y}{y} \right| \approx \text{Cond}(f; x) \left| \frac{\delta x}{x} \right|
\]
where the \textit{CONDITION NUMBER} is
\[
\text{Cond}(f; x) = \left| \frac{xf'(x)}{f(x)} \right|
\]
It is clear from this definition then that the condition number not only depends on the function \( f \) itself, but also it depends on the value(s) of \( x \) which are used as input to the black box \( f \).

\[\square\]

\underline{Example}

Consider the function \( y = f(x) = 1/x \). The derivative of this function is
\[
f'(x) = -\frac{1}{x^2}.
\]

From this we can calculate the condition number of the function
\[
\text{Cond}(f; x) = \left| \frac{xf'(x)}{f(x)} \right| = \left| \frac{x \cdot -\frac{1}{x^2}}{\frac{1}{x}} \right| = 1
\]
So for this function \( f(x) = \frac{1}{x} \) the condition number is independent of the value of \( x \). This may seem strange as we know that \( f(x) \) ‘blows up’ near zero but it is important to remember that we are dealing with relative error \( \frac{\delta x}{x} \) which could be large if \( x \) is sufficiently small.

Now let’s consider the function
\[
y = g(x) = \sin(x)
\]
Figure 12: Condition number plotted against $x$ for $\sin(x)$
which has the derivative
\[ g'(x) = \cos(x). \]

The condition number is
\[
\text{Cond}(g, x) = \frac{|x \cos(x)|}{|\sin(x)|} = \frac{|x|}{|\tan(x)|}.
\]

(5)

In this case the condition number is dependent on the value of \( x \). Figure 12 shows the condition number plotted against \( x \). For small values of \( x \) (e.g. \( x \in \left[ -\frac{3\pi}{4}, \frac{3\pi}{4} \right] \)) the condition number is reasonable. However, the condition number goes to infinity as \( x \) approaches \( \pm \pi \). This is because \( \sin(\pm \pi) = 0 \) so the values of \( y \) will be very small giving the potential for large relative errors in \( y \). It seems strange that we do not get the condition number going to infinity for \( x \) close to zero as well because \( \sin(0) = 0 \). But because both \( x \) and \( y \) are small at this point, their relative errors are comparable.

Now let’s look at a polynomial function
\[ h(x) = 2x^2 + x - 1 \]

which has the following derivative
\[ h'(x) = 4x + 1. \]

This gives the condition number
\[
\text{Cond}(h; x) = \left| \frac{xh'(x)}{h(x)} \right| = \left| \frac{x(4x + 1)}{2x^2 + x - 1} \right| = \left| \frac{4x^2 + x}{2x^2 + x - 1} \right|.
\]

Again, this condition number is dependent on \( x \). Figure 12 shows the condition number plotted against \( x \). For values when \( h(x) \approx 0 \) (\( x \) close to \(-1\) and \( \frac{1}{2} \)) we have a very large condition number. For \( x \) close to zero, however, the condition number is close to one and for large values of \( x \) the condition number is close to two. To understand why the condition number is close to
2 for large $x$ consider the function $k(x) = x^n$. Its derivative is $k'(x) = nx^{n-1}$ giving a condition number of

$$
\text{Cond}(k; x) = \left| \frac{x \cdot nx^{n-1}}{x^n} \right| = |n|
$$

Place here the stuff for more advanced students that works out the $m \times n$ dimensional case. Also work out the estimate when there are roundoff errors.

We will revisit the condition number concept once more when we discuss the solution of linear algebraic equations. **Note**: some software will report a sensitivity of the calculation. But be very aware of how this number is defined (or as is the case in Lapack and others, that they do not report the condition number as defined above, but the inverse of it.)
The idea of perturbing the input and looking at the outcome in the output is called *forward perturbation (error) propagation*. In some cases it can lead to an overestimate of the error encountered in the practical setting. Why this is the case is beyond the scope of these notes. We will limit ourselves to stating that there is an alternative approach to sensitivity analysis, called *backward error analysis*. See Figure 14. In backward error propagation we ask: How much error in input would be required to explain all output error? Backward error propagation assumes that approximate solution to problem is good IF IT IS THE exact solution to a “nearby” problem.

**Example** Want to approximate \( f(x) = e^x \). We evaluate its accuracy at \( x = 1 \).

\[
\begin{align*}
  f(x) &= e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 \cdots \\
  \hat{f}(x) &= 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3
\end{align*}
\]
Forward error $= \hat{f}(x) - f(x)$

Backward Error: 1) Find $\hat{x}$ such that $f(\hat{x}) = \hat{f}(x)$

\[
f(\hat{x}) = e^{\hat{x}} \quad \therefore \quad \hat{x} = \log(f(\hat{x})) = \log(\hat{f}(x))
\]

\[
f(x = 1) = 2.718282 \quad \hat{f}(x) = 2.666667, \text{ to seventh decimal place.}
\]

\[
\hat{x} = \log(2.666667) = 0.980829
\]

The following are different and cannot be compared:

Forward error $\hat{f}(x) - f(x) = -0.051615$ (OK)

Backward error $\hat{x} - x = -0.019171$

(Ok, because output is correct for a slightly perturbed input.)

An algorithm is stable if small perturbations to the input lead to small changes in the output. Using backward error analysis then the algorithm is stable if the result it produces is the exact solution to a nearby problem. If algorithm is stable the output is exact for a nearby problem but solution to nearby problem is not necessarily close to the solution to the original problem UNLESS problem is stable. Therefore, inaccuracy can result from applying a stable algorithm to an ill-conditioned problem as well as from applying an unstable algorithm to a well-conditioned problem.

\[
\begin{align*}
\text{5} & \quad \text{SOLUTION OF ALGEBRAIC EQUATIONS} \\
\end{align*}
\]

We are going to focus on the case when $f(x) = 0$ is a single equation. This problem is interesting in its own right, but understanding how to solve the
single equation problem allows us to appreciate the challenge when \( f(x) = 0 \) is an \( n \)--dimensional problem. In the case of the single equation it might be possible to already have some ideas as to whether roots exist, how many, whether they are complex (a good inference is obtained by just plotting the equation, for example). In fact, it is always a good idea to have some ideas as to where the roots lie, if they exist. In the \( n \)--dimensional case, it might not be at all possible to even know what the set of equations \( f(x) = 0 \) look like.

We focus in this section on the solution of the scalar problem. The linear and nonlinear \( n \)--dimensional problem will be considered after we introduce some basic linear algebra.

Problem Statement:

*Find the root(s) \( x \) (possibly complex valued) such that \( f(x) = 0 \) \( f : R \rightarrow R \cap C^1 \) \( x \in R \).*

In most cases it is not possible to solve this problem analytically; for polynomials, for example, analytical formulas exist when the polynomial has a non-negative integer degree less than 5. We’ll be discussing polynomial root finding here as well, but first we consider the general case when \( f \) is not necessarily a polynomial.

Many root-finding methods are iterative and require some initial guess(es):

* Two-Point Methods, require 2 guesses. Two of these are the bisection method and the secant method.

* One Point Methods, require a single guess. Two of these are regula falsi and Newton-Raphson.

Whether it uses 1 or 2 guesses is less important than whether it is fast, well-conditioned, and appropriate for the problem at hand.

Of the methods we will look at, the bisection method is important because, although really slow and clumsy on high dimensional problems, it is useful when there is no explicit function; it may be a set of rules.
The Newton-Raphson is a second order method, and thus the fastest of the methods considered here. However, it requires explicit knowledge of the derivative of $f(x)$. The Secant method is super linear, so slower than Newton-Raphson. It is useful when the derivative of $f(x)$ is not explicitly known.

The fixed-point method (or functional iteration) is a simple first order method. It is slow, but is trivial to implement. However, perhaps the most useful thing about this method is that it teaches us a lot about how and why methods work or fail to converge, how to modify methods to make them more well-conditioned, and whether a root exists and is unique.

5.1 Bisection Method

This is also known as a bracketing method. The general strategy in bracketing is the following:

If $f \in C[a,b]$ and $f(a)f(b) \leq 0$ then by the intermediate value theorem $\exists$ at least one zero in $[a,b]$. Condition $f(a)f(b) \leq 0$ means that $f(a)$ and $f(b)$ are opposite in sign. So choose $a,b$ to create a bracket containing the root you want $f(\alpha_1) = 0$. See Figure 15 If the interval contains the root, divide it and search in the subinterval that contains the root. This test-and-divide strategy is iterated making the brackets become smaller. The process continues until the root is found, to within some tolerance.

5.1.1 The Basic Bisection Algorithm

Given $f(x), a,$ and $b$:

1. Is $f(a)f(b) < 0$? if so, set

   $c = \frac{1}{2}(a + b)$.

2. Is $f(a)f(c) < 0$? (else $f(c)f(b) < 0$)

   \[
   \begin{cases} 
   b = c & \text{if } (f(a)f(c)) < 0 \\
   a = c & \text{else } a = c \\
   f(c) = 0 & \text{else } f(c) = 0 
   \end{cases}
   \]
3. Repeat

Implementation Details. It is unlikely that the above algorithm would work on a computer. Consider the following issues and their resolution:

- Due to roundoff, unlikely to find $f(c) = 0$ exactly. So we use $|f(c)| < |\delta|, \delta \ll 1$ instead.

- Potential round-off problem: Consider the following sequence of events:
  \[
  c_0 = \frac{a + b}{2} \\
  c_1 = \frac{c_0 + b}{2} = \frac{1}{4}(a + 3b) \\
  c_2 = \frac{1}{2\cdot4}(a + 7b) \\
  c_3 = \frac{1}{2\cdot4\cdot8}(a + 15b)
  \]
  Could otherwise lead to $c \not\in [a, b]$. But generally, one can see that the first term on the right can become so small compared to the second term
on the right. This can lead to errors, due to roundoff. In general, it is always better to add a small correction to a previous approximation. The better implementation: use
\[ c = a + \frac{(b - c)}{2} \]
in the code, instead of
\[ c = \frac{a + b}{2}. \]

- We really only care about sign of \( f(a)f(b) \) therefore to avoid underflows and overflows compute instead \( sgn(f(a))sgn(f(b)) \). See Figure 16.

- You also don’t want to get into an infinite loop because the user might have entered incorrect inputs. Use a stopping criteria, such as a maximum number of bisections to avoid runaway jobs.

So in the implementation you need to have three stopping criteria:

1. maximum number of bisections (avoids runaway jobs)
2. location error: \( |b - a| < \epsilon \)
3. accuracy of root: \( |f(c)| < \delta \)
The code is written so that all three criteria must be satisfied.

*Could you make a drawing of a function that will fail one of the above criteria but not the other one?*

Error Analysis: Assume that we have the following brackets: \([a_0, b_0], [a_1, b_1]\) \cdots. Also, assume that \(a_0 \leq a_1 \leq a_2 \cdots \leq b_0\), \(a_i\) is a sequence that increases and bounded by \(b_0\) from above. Also, \(b_0 \geq b_1 \geq b_2 \cdots \geq a_0\), \(b_i\) is a sequence that decreases and bounded by \(a_0\) from below.

Then

\[
\begin{align*}
  b_{n+1} - a_{n+1} &= \frac{1}{2}(b_n - a_n) \quad n \geq 0 \\
  b_1 - a_1 &= \frac{1}{2}(b_0 - a_0) \\
  b_2 - a_2 &= \frac{1}{2}(b_1 - a_1) = \frac{1}{2} \cdot \frac{1}{2}(b_0 - a_0) \\
  &\vdots \\
  b_n - a_n &= 2^{-n}(b_0 - a_0)
\end{align*}
\]

Then \(\lim_{n \to \infty} b_n - \lim_{n \to \infty} a_n = \lim_{n \to \infty} 2^{-n}(b_0 - a_0) = 0\). Note that \(r = \lim_{n \to \infty} a_n = \lim_{n \to \infty} b_n\) therefore \(0 \geq f(\lim_{n \to \infty} a_n) f(\lim_{n \to \infty} b_n) = f(r)f(r)\). Hence \(f(r) = 0\).

Suppose we are in a certain bracket \(n\): \([a_n, b_n]\). What’s the best estimate of the location of the root? The midpoint is sensible: \(c_n = \frac{a_n + b_n}{2}\), then

\[
\begin{align*}
  |r - c_n| &\leq \frac{1}{2}(b_n - a_n) = 2^{-(n+1)}(b_0 - a_0)
\end{align*}
\]

**Theorem:** Bisection Method: If \([a_0, b_0], [a_1, a_2] \cdots [a_n, b_n] \cdots\) denote the intervals in the bisection method, then the \(\lim_{n \to \infty} a_n\) and \(\lim_{n \to \infty} b_n\) exist, are equal and represent zero of \(f\). If \(r = \lim_{n \to \infty} \frac{1}{2}(a_n + b_n)\) then \(|r - c_n| \leq 2^{-(n+1)}(b_0 - a_0)\).
Example Suppose \([50, 63]\). How many steps should be taken to compute a root with relative accuracy of 1 part in \(10^{-12}\)?

\[
\frac{|r - c_n|}{|r|} \leq 10^{-12}.
\]

We know that \(r \geq 50\) therefore \(\frac{|r - c_n|}{x_0} \leq 10^{-12}\)

By the above theorem:

\[
|r - c_n| \leq 2^{-(n+1)}(b_0 - a_0) = 2^{-(n+1)}(63 - 50)
\]

\[
\frac{|r - c_n|}{50} \leq \frac{2^{-(n+1)}13}{50} \leq 10^{-12}.
\]

Solving for the inequality, we have that \(n \geq 37\).

5.2 Regula-Falsi Method (False-Position Method)

The bisection method does not use the actual values of \(f(x)\). It only uses their sign. However, the values could be exploited.

In what follows, assume \(f(x)\) continuous such that \(f(a)f(b) < 0\). One way to use values of \(f(x)\) is to bias the search according to value of \(f(x)\): use a weighted average.

Choose \(c\) as the intercept of the secant line through the points \((a, f(a))\) and \((b, f(b))\):

\[
\frac{y - f(b)}{x - b} = \frac{f(a) - f(b)}{a - b}.
\]

This gives the formula for the secant line.

Pick the intercept \(y = 0\), then the first approximation is

\[
x_1 = \frac{af(b) - bf(a)}{f(b) - f(a)}.
\]

\(x_1\) is an approximation to \(x^*\), the root. (The algorithm is depicted in Figure 17.)
Figure 17: First few iterates of the Regula-Falsi Algorithm.
As in the Bisection Method, if \( f(x_1) \neq 0 \Rightarrow f(a)f(x_1) < 0 \) or \( f(x_1)f(b) < 0 \) and there must be a root \( x^* \in [x_1, b] \). Suppose \( f(x_1)f(b) < 0 \), then

\[
x_2 = \frac{x_1 f(b) - b f(x_1)}{f(b) - f(x_1)}.
\]

Same process for \( x_3 \), and so on.

This is the pseudo code for the Regula Falsi method:

User inputs: \( a,b \), max, epsilon
Initialize: iteration = 0
\[
xold = b \quad // to start out
\]

Inside a loop with condition that iteration <=max:

\[
\text{while(iteration} \leq \text{max)}
\]
compute \( f(a) \)
compute \( f(b) \)
\[
x = a - \frac{(f(a)*(b-a))}{(f(b)-f(a))} / (f(b)-f(a))
\]
increment iteration

Test:
if \( |x-xold| < \epsilon \text{selon}*|x| \) then
output \( x \)
goto stop
else
output (iteration, a, b, x)
xold = x
compute \( f(x) \)

if \( f(a)*f(x) > 0 \) // here we are testing // for a positive sign, // not necessarily the value
\[
a = x
\]
else
\[
b = x
\]
A potential problem: what should be the stopping criteria (Recall that in bisection we had intervals getting smaller)?

We could use

\[(6) \quad |x_n - x_{n-1}| < \varepsilon\]

and

\[(7) \quad |f(x_n)| < \delta.\]

However these can lead to some problems:

Suppose \(|x_n - x_{n-1}| = \frac{1}{n}\), which is satisfied by (6) when \(n \geq \frac{1}{\varepsilon}\). The problem will be that \{\(x_n\)\} will actually diverge.

Suppose \(f(x) = (2-x)^7\), \(x^* = 2\) and \(x_n = 2 - \frac{1}{n} \Rightarrow |f(x_n)| < 10^{-2} \forall n > 1\)

whereas \(|x^* - x_n| < 10^{-2}\) only for \(n > 100\)

So the test will signal convergence prematurely!

Instead the stopping criteria should be

\[|x_n - x_{n-1}| < \varepsilon |x_n|\]

and use a relative error as well for \(f(x_n)\).

\[\square\]

5.3 Newton Raphson Method (NR Method)

A second order method, useful when we know the derivative of \(f(x)\) explicitly.

Problem Statement: Find \(x = p\) such that \(f(x) = 0\)
Suppose \( f \in C^2[a, b] \). Let \( \bar{x} \in [a, b] \) be an approximation to \( p \) such that \( f'(\bar{x}) \neq 0 \) and \( |\bar{x} - p| \) is “small.” Then

\[
f(x) = f(\bar{x}) + (x - \bar{x})f'(\bar{x}) + \frac{(x - \bar{x})^2}{2}f''(\xi(x)) + O(|x - \bar{x}|^3)
\]

where \( \xi(x) \) lies between \( x \) and \( \bar{x} \). Since \( f(p) = 0 \), with \( x = p \), (8) gives

\[
0 = f(\bar{x}) + (p - \bar{x})f'(x) + \frac{(p - \bar{x})^2}{2}f''(\xi(p)).
\]

If \( |p - \bar{x}| \) is small, then \( |p - \bar{x}|^2 \) smaller, and

\[
0 \approx f(\bar{x}) + (p - \bar{x})f'(\bar{x})
\]

is a good approximation. Solving for \( p \):

\[
p \approx \bar{x} - \frac{f(\bar{x})}{f'(\bar{x})}
\]

The NR method begins with an estimate \( p_0 \) and generates a sequence \( \{p_n\} \),

\[
p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad n \geq 1, \quad p_0 \equiv \text{initial guess}
\]

See Figure 18 for an illustration of the first steps of the algorithm. **Pros:** Faster than the bisection and regula-falsi method (which are linear), and faster than the secant method (see 5.5) (which is superlinear). **NR Method has quadratic convergence.**

**Cons:** a) Need \( f'(x) \) or a good approximation to it (in general an approximations will produce less than quadratic convergence).

b) Not guaranteed to converge unconditionally.

**Newton-Raphson Algorithm** The algorithm calls a function called \( f \) which delivers \( f(x) \) given \( x \), and a function called \( f' \) which delivers the derivative of \( f \), given \( x \).

Inputs: initial guess \( p_0 \), tolerance TOL, maximum iterations \( N_0 \),

72
Figure 18: First few steps of the Newton Raphson (NR) iteration process.
Set \( i = 1 \)
While \( i \leq N_0 \) {
   \[ p = p_0 - f(p_0)/f'(p_0), \quad \% \text{to compute } p_i \]
   If \(|p-p_0| < TOL\) then
      OUTPUT \( p \)
      STOP
   else
      \( i = i + 1 \)
      \( p_0 = p \) \quad \% \text{update } p_0
   end if
}
Output (‘Method failed after \( N_0 \) iterates’)
STOP

Error Analysis Let the error \( e_n \equiv p_n - p \). For simplicity we assume no round-off errors in this analysis.

Assume \( p \) is a simple zero contained in \( x \in [a,b] \) and \( f' \) continuous for \( x \in (a,b) \). Then

\[
e_{n+1} = p_{n+1} - p = p_n - \frac{f(p_n)}{f'(p_n)} - p
\]

\[
= e_n - \frac{f(p_n)}{f'(p_n)} = \frac{e_n f'(p_n) - f(p_n)}{f'(p_n)}
\]

By Taylor’s Theorem:

\[
0 = f(p) = f(p_n - e_n) = f(p_n) - e_n f'(p_n) + \frac{1}{2} e_n^2 f''(\xi_n)
\]

where \( \xi_n \) is some number between \( p_n \) and \( p \). Rearranging terms we see that

\[
e_n f'(p_n) - f(p_n) = \frac{1}{2} f''(\xi_n) e_n^2
\]

Which when substituted into (10) yields

\[
e_{n+1} = \frac{1}{2} f''(\xi_n) e_n^2 \approx \frac{1}{2} f''(p) e_n^2 = Ce_n^2
\]
Therefore, the algorithm has at least quadratic convergence rate.

Note that we haven’t established convergence itself, just the rate. By (11), the proof is as follows: if \( e_n \) small and if \( \frac{1}{2} \frac{f''(\xi_n)}{f'(p_n)} \) is not too large \( \Rightarrow e_{n+1} \) will be smaller than \( e_n \).

Define \( c(\delta) := \frac{1}{2} \max_{|x-p| \leq \delta} |f''(x)|/\min_{|x-p| \leq \delta} |f'(x)|, \ \delta > 0. \)

Select \( \delta \) small enough, so that \( \delta c(\delta) < 1 \). This is possible because as \( \delta \to 0 \), then \( c(\delta) \) converges to \( \frac{1}{2} \frac{|f''(p)|}{|f'(p)|} \), and \( \delta c(\delta) \) converges to 0.

Having fixed \( \delta \), set \( \rho = \delta c(\delta) \).

Suppose we start the NR Method with \( p_0 \) satisfying \( |p_0 - p| \leq \delta \). Then \( |e_0| \leq \delta \) and \( |\xi_0 - p| \leq \delta \) thus

\[
\frac{1}{2} |f''(\xi_0)|/|f'(p_0)| \leq c(\delta),
\]

by the definition of \( c(\delta) \).

Taking (11) \( |p_1 - p| = |e_1| \leq e_0^2 c(\delta) = |e_0||e_0|c(\delta) \leq |e_0|\delta c(\delta) = |e_0|\rho < |e_0| \leq \delta \), Therefore \( p_1 \) lies within \( \delta \) distance to \( p \). Repeating,

\[
\begin{align*}
|e_1| & \leq \rho |e_0| \\
|e_2| & \leq \rho |e_1| \leq \rho^2 |e_0| \\
|e_3| & \leq \rho^3 |e_0| \\
& \vdots \\
|e_n| & \leq \rho^n |e_0|
\end{align*}
\]

Since \( 0 \leq \rho < 1 \) then \( \lim_{n \to \infty} \rho^n = 0 \). Hence, \( \lim_{n \to \infty} e_n = 0 \)

\[\square\]

**Theorem: (Newton Raphson)** Let \( f \in C^2[a,b] \). Assume \( p \in [a,b] \) is a simple zero (a root \( p \) such that \( f(p) = 0 \) and \( f'(p) \neq 0 \)). Then there exists a neighborhood of \( p \) and a constant \( C \) such that if the Newton Raphson method, started in that neighborhood, yields successive guesses that become closer to \( p \) and satisfy

\[
|p_{n+1} - p| \leq C(p_n - p)^2 \quad n \geq 0.
\]
In some situations NR is guaranteed to converge from any arbitrary starting point:

**Theorem:** If \( f \in C^2(R) \), is an increasing, convex function, and \( f(p) = 0 \) then \( p \) is unique and the Newton method will converge to it from any starting point.

**Exercise:** Prove Theorem 2. Hints: Convex means \( f''(x) > 0 \forall x \). Increasing means: \( f'(x) > 0 \). You may also use (12).

**Example** An efficient method for computing the square root of a number \( R \):

Let \( R > 0 \) and \( x = \sqrt{R} \). then \( x^2 - R = 0 \). Then Newton Raphson formula yields \( x_{n+1} = \frac{1}{2} \left( x_n + \frac{R}{x_n} \right) \) (credited to Heron, Greek Engineer circa 100 B.C. - 100 A.D).

5.4 Steffensen Method

A variant of the NR Method, useful when \( f'(x) \) is not known explicitly.

The algorithm is: with an initial guess \( p_0 \),

\[
p_{n+1} = p_n - \frac{f(p_n)}{g(p_n)}, \quad n = 0, 1, 2, \ldots,
\]

where \( g(x) = \frac{[f(x+f(x))-f(x)]}{f(x)} \).

It is an exercise to show how and to what extent \( g(x) \) is an approximation to \( f'(x) \). To prove at least quadratic convergence we require that \( g \in C^3[a, b] \).
5.5 The Secant Method

If you approximate $f'(x)$ by a secant in the NR, you obtain the Secant Method.

So, unlike Newton’s Method which requires $f'(x)$, you do not need it here.

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

since $f'(x) = \lim_{n \to x} \frac{f(h) - f(x)}{h - x}$

The method is thus

$$p_{n+1} = p_n - f(p_n) \left[ \frac{p_n - p_{n-1}}{f(p_n) - f(p_{n-1})} \right] \quad n \geq 1$$

We need two starting values $p_0$ and $p_1$.

The Rate of Convergence for Secant:

Stated without proof:

$$e_{n+1} = C e_n e_{n-1} \sim A |e_n|^{(1+\sqrt{5})/2}$$

Since this is the golden mean $\frac{1 + \sqrt{5}}{2} \approx 1.62 < 2$ the convergence rate is superlinear, i.e. better than linear, but not as good as quadratic convergence rate.

The first few steps of the algorithm for the Secant method are depicted in Figure 19.

**Secant Method Algorithm:**

Find $f(x) = 0$ given $p_0, p_1$ initial guesses. Require a function which if given $x$ will produce $f(x)$.

Input: $p_0, p_1$ tolerance TOL, and a maximum number of iterations $N_0$. The output is $p$, or a diagnostic message.
Figure 19: First few steps of the Secant Algorithm.
\[
q_0 = f(p_0), \quad q_1 = f(p_1)
\]

While \( i \leq N_0 \) {
\[
p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0) \quad \% \text{compute } p_i
\]
if \(|p - p_1| < \text{TOL}\) then
\[
\text{output } p \quad \% \text{done}
\]
STOP
end if
\[
i = i + 1
\]
\[
p_0 = p_1 \quad \% \text{update } p_0, q_0, p_1, q_1
\]
\[
q_0 = q_1
\]
\[
p_1 = p
\]
\[
q_1 = f(p)
\]
}
Output (‘Method failed after \(N_0\) iterates’).
STOP

---

### 5.6 Fixed Point Iteration

**definition:** A fixed point \( p \) is the value \( p \) such that

\[
g(p) = p.
\]

Fixed Point problems and root-finding problems \( f(x) = 0 \) are equivalent: Let \( g(p) - p = f(p) \Rightarrow f(p) = 0 \). Hence, if a function has a fixed point \( g(p) = p \) then \( f(p) = 0 \), i.e., \( f \) has a zero at \( p \).

Three Problems:

1. Which functions will have a fixed point?
2. How do we determine the fixed point?
3. Is the fixed point unique?
Example $g(x) = x$, $0 \leq x \leq 1$ has a fixed point at each $x \in [0,1]$. To see this, just plot it.

Example $g(x) = x^2$ has 2 fixed points in $[0,2]$, at $x = 0$ and $x = 1$. □

**Theorem** (Existence and Uniqueness): If $g \in C[a,b]$ such that $g(x) \in [a,b]$ $\forall x \in [a,b]$ then $g(x)$ has a fixed point in $[a,b]$.

Suppose, in addition, that $g'(x)$ exists on $(a,b)$ and that a positive constant $k < 1$ exists with

$$|g'(x)| \leq k < 1 \quad \forall x \in (a,b)$$

then the fixed point in $[a,b]$ is unique.

**Proof:**

*Existence:* If $g(a) = a$ or $g(b) = b$, then existence of fixed point is clear. Suppose not, then it must be true that $g(a) > a$ and $g(b) < b$. Define $h(x) = g(x) - x$. Then $h$ is continuous on $[a,b]$ and

$$h(a) = g(a) - a > 0 \quad h(b) = g(b) - b < 0$$

The Intermediate Value Theorem implies that there exists $p \in (a,b)$ for which $h(p) = 0$. Thus, $g(p) - p = 0$ implies $p$ is fixed point of $g$.

*Uniqueness:* Suppose, in addition $|g'(x)| \leq k < 1 \quad \forall x \in (a,b)$ and that $p$ and $q$ are both fixed points in $[a,b]$ with $p \neq q$. By the Mean Value Theorem a number exists between $p$ and $q$ such that

$$\frac{g(p) - g(q)}{p - q} = g'(\xi),$$

for some $\xi \in (a,b)$. Then $|p - q| = |g(p) - g(q)| = |g'(\xi)| |p - q| \leq k |p - q| < |p - q|$, which is a contradiction.

This contradiction comes from the statement $p \neq q$ therefore $p = q$ and the fixed point is unique. □

**Fixed-Point Iteration** Here we employ an iterative procedure to find the solution of $g(p) = p$. Questions that naturally arise in connection with this problem are: (1) does the iterative procedure converge to the exact answer? (2) under what conditions does it converge?, (3) at what rate?
The iteration is rather straightforward: Pick a \( p_0 \) and generate a sequence \( \{p_n\}_{n=0}^{\infty} \) such that \( p_n = g(p_{n-1}) \) for \( n \geq 1 \). If the sequence converges to \( p \) and assuming \( g \) is continuous, by the theorem above we have:

\[
p = \lim_{n \to \infty} p_n = \lim_{n \to \infty} g(p_{n-1}) = g(\lim_{n \to \infty} p_{n-1}) = g(p).
\]

The algorithm is depicted in Figure 20

**Fixed Point Algorithm**

Required, a function \( g \) such that if provided with \( x \) will yield \( g(x) \).

Input: initial guess \( p_0 \), TOL absolute error tolerance, \( N_0 \), the maximum number of iterations allowed. Output: the fixed point \( p \) or a message of failure

\( i=1 \)
While $i \leq N_0$
\[
    p = g(p_0) \quad \text{% Compute } p_i
\]
if $|p-p_0| < \text{TOL}$ then
\[
    \text{output } p \quad \text{% found fixed point, to within TOL}
\]
Stop
\[
    i = i + 1
\]
$p_0 = p \quad \text{%Update } p$
\[
}\}
\]
Output ('Max iterations exceeded, $i > N_0$')
END

\[2\]

Theorem: (Fixed Point Iteration) Let $g \in C[a, b]$ and suppose $g(x) \in [a, b] \forall x \in [a, b]$. Suppose in addition that $g'$ is continuous on $(a, b)$ with
\[
    |g'(x)| \leq k < 1 \quad \forall x \in (a, b).
\]
If $g'(p) \neq 0$, then for any $p_0$ in $[a, b]$, the sequence
\[
    p_n = g(p_{n-1}), \quad \text{for } n \geq 1,
\]
converges only linearly to the unique fixed point in $[a, b]$.

Proof: The Fixed Point Theorem says $\{p_n\}_{n=0}^{\infty} \to p$. Since $g'$ exists on $(a, b)$ we can apply the Mean Value Theorem to $g$ to show that for any $n$
\[
    p_{n+1} - p = g(p_n) - g(p) = g'(\xi_n)(p_n - p)
\]
where $\xi_n$ lies between $p_n$ and $p$. Since $\{p_n\}_{n=0}^{\infty} \to p$, and $\{\xi_n\}_{n=0}^{\infty} \to p$.

Since $g'$ is continuous on $(a, b)$ we have
\[
    \lim_{n \to \infty} g'(\xi_n) = g'(p).
\]
Thus
\[
    \lim_{n \to \infty} \frac{p_{n+1} - p}{p_n - p} = \lim_{n \to \infty} g'(\xi_n) = g'(p) \text{ and }
\]
\[
    \lim_{n \to \infty} \frac{|p_{n+1} - p|}{|p_n - p|} = |g'(p)|
\]
82
therefore, the fixed point iteration converges linearly if $g'(p) \neq 0$.

\[ \Box \]

Remark: Under certain circumstances we can get higher-order convergence, i.e. when $g'(p) = 0$. The following theorem addresses this situation:

**Theorem:** Suppose $p$ is a solution of $x = g(x)$. Assume as well that $g'(p) = 0$ and $g''$ continuous and strictly bounded by $M$ on an open interval $I$ containing $p$. Then $\exists \delta > 0$ such that $p_0 \in [p - \delta, p + \delta]$. Then the sequence $p_n = g(p_{n-1})$, $n \geq 1$ will converge quadratically:

\[ |p_{n+1} - p| < \frac{M}{2} |p_n - p|^2 \]

\[ \Box \]

**Fixed Point Iteration:** Let $g \in C \subseteq [a, b]$ and suppose $g(x) \in [a, b] \forall x \in [a, b]$. Suppose in addition that $g'$ exists on $(a, b)$ with $k > 0$ and constant such that

\[(13) \quad |g'(x)| \leq k < 1 \quad \forall x \in (a, b).\]

If $p_0$ is any number in $[a, b]$, then

\[ p_n = g(p_{n-1}), \quad n \geq 1 \]

converges to unique fixed point in $[a, b]$.

**Proof:** From Fixed Point Theorem, a unique fixed point exists in $[a, b]$. Since $g$ maps $[a, b]$ into itself, the sequence $\{p_n\}_{n=0}^{\infty}$ is defined $\forall n \geq 0$ and $p_n \in [a, b] \quad \forall n$.

Using (13) and the Mean Value Theorem

\[ |p_n - p| \approx |g(p_{n-1}) - g(p)| = |g'(\xi)||p_{n-1} - p| \leq k|p_{n-1} - p| \]

where $\xi \in (a, b)$.

By induction

\[(14) \quad |p_n - p| \leq k|p_{n-1} - p| \leq k^2|p_{n-2} - p| \leq \cdots k^n|p_0 - p|.\]

Since $k < 1$

\[ \lim_{n \to \infty} |p_n - p| \leq \lim_{n \to \infty} k^n|p_0 - p| = 0 \]

83
and
\[ \{p_n\}_{n=0}^{\infty} \to p. \]

Corollary If \( g \) satisfies the hypothesis of the Fixed Point Iteration theorem, bounds for the error involved in using \( p_n \) to approximate \( p \) are given by
\[
|p - p_n| \leq k^n \max \{ p_0 - a, b - p_0 \}, \quad (a)
\]
and
\[
|p - p_n| \leq \frac{k^n}{1 - k} |p_1 - p_0|, \quad \text{for } n \geq 1. \quad (b)
\]

Proof: (a) follows from (14):
\[
|p_n - p| \leq k^n |p_0 - p| \leq k^n \max \{ p_0 - a, b - p_0 \} \text{ since } p \in [a, b].
\]

For \( n \geq 1 \)
\[
|p_{n+1} - p_n| = |g(p_n) - g(p_{n-1})| \leq k|p_n - p_{n-1}| \leq \ldots k^n |p_1 - p_0|.
\]

Therefore, for \( m > n \geq 1 \),
\[
|p_m - p_n| = |p_m - p_{m-1} + p_{m-1} - p_{m-2} + \ldots + p_{n+1} - p_n|
\leq |p_m - p_{m-1}| + |p_{m-1} - p_{m-2}| + \ldots + |p_{n+1} - p_n|
\leq k^{m-1} |p_1 - p_0| + k^{m-2} |p_1 - p_0| + \ldots + k^n |p_1 - p_0|
= k^n (1 + k + k^2 + \ldots + k^{m-n-1}) |p_1 - p_0|.
\]

Since \( \lim_{m \to \infty} p_m = p \) then
\[
|p - p_n| = \lim_{m \to \infty} |p_m - p_n| \leq k^n |p_1 - p_0| \sum_{i=0}^{\infty} k^i
\]

Since \( \sum_{i=0}^{\infty} k^i = \frac{1}{1-k} \), for \( k < 1 \), then
\[
|p - p_n| \leq \frac{k^n}{1 - k} |p_1 - p_0|.
\]
Figure 21: Plot of $g(x)$.

Remark: The rate at which $\{p_n\}$ converges depends on $k$: the smaller the value of $k$, the faster it will converge.

Example: Consider $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$. This function is illustrated in Figure 21$^4$.

First we wish to ensure that the function maps $[0, \infty)$ into itself.

$$g([0, \infty)) = [18^{\frac{1}{3}}, \infty) \subset [0, \infty)$$

Next we look at the derivative of $g(x)$

$$g'(x) = \frac{1}{3} (3x + 18)^{\frac{2}{3}} \times 3$$

$$= \frac{1}{(3x + 18)^{\frac{2}{3}}}$$

$^4$Get matlab code used in the example from www.physics.arizona.edu/ restrepo/475A/Notes/code2.m
\[ |g'(x)| = \frac{1}{(3x + 18)^{\frac{2}{3}}} \leq \frac{1}{18^{\frac{2}{3}}} < 1 \text{ for } x \in [0, \infty) \]

This fulfills the requirements for a unique fixed point to exist in \([0, \infty)\). It also ensures that if we start with any non-negative value of \(x\) we will converge to the fixed point. The table below shows the first ten iterations for three different values of \(x_0\). Figure 22a and Figure 22b illustrate the iteration history and the logarithm of the error, for the case starting with \(x_0 = 0\). Figure 23a and Figure 23b illustrate the iteration history and the logarithm of the error, for a case starting with \(x_0 = 100\). Finally, Figure 24a and Figure 24b illustrate the iteration history and the logarithm of the error, for a case starting with \(x_0 = 10000\).

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>0</th>
<th>100</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>2.6207</td>
<td>6.8256</td>
<td>31.0785</td>
</tr>
<tr>
<td>(x_2)</td>
<td>2.9573</td>
<td>3.3760</td>
<td>4.8093</td>
</tr>
<tr>
<td>(x_3)</td>
<td>2.9952</td>
<td>3.0412</td>
<td>3.1889</td>
</tr>
<tr>
<td>(x_4)</td>
<td>2.9995</td>
<td>3.0046</td>
<td>3.0208</td>
</tr>
<tr>
<td>(x_5)</td>
<td>2.9999</td>
<td>3.0005</td>
<td>3.0023</td>
</tr>
<tr>
<td>(x_6)</td>
<td>3.0000</td>
<td>3.0001</td>
<td>3.0002</td>
</tr>
<tr>
<td>(x_7)</td>
<td>3.0000</td>
<td>3.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>(x_8)</td>
<td>3.0000</td>
<td>3.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>(x_9)</td>
<td>3.0000</td>
<td>3.0000</td>
<td>3.0000</td>
</tr>
<tr>
<td>(x_{10})</td>
<td>3.0000</td>
<td>3.0000</td>
<td>3.0000</td>
</tr>
</tbody>
</table>

The iterations for the three different starting points all appear to converge to the point 3. The log error plots are straight lines and they all have the same slope, indicating the same rate of convergence.

We can also prove analytically that 3 is the fixed point of \(g(x)\). A fixed point of \(g(x)\) satisfies

\[ x = (3x + 18)^\frac{1}{3}. \]

We can rearrange this to get \(x^3 - 3x - 18 = 0\) which has one real root, \(x = 3\).
Figure 22: *Iteration history and the logarithm of the error. Case starting with* $x_0 = 0$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$. 
Figure 23: *Iteration history and the logarithm of the error. Case starting with* $x_0 = 100$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$. 
Figure 24: *Iteration history and the logarithm of the error. Case starting with* $x_0 = 10000$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$. 
**Example:** Consider the function \( g(x; \alpha) = \frac{x^2 - 2x + 1}{\alpha} \) for \( x \in [0, 2] \). We will start with the initial value \( x_0 = 0.1 \) and consider what happens for various values of \( \alpha \).

Figure 25–Figure 28 show the iterations for \( \alpha = 2.5, 1.5, 0.5 \) and 0.49, respectively. They also plot \( g(x; \alpha) \) on the same graph as \( y = x \) so we can see the fixed point, and finally plot the log of the error for each value of \( \alpha \). We can see that for both \( \alpha = 2.5 \) and \( \alpha = 1.5 \) the iteration converges to a fixed point. The log error plots are straight lines with the slope showing the convergence rate (Question: Why does the log error plot flatten off for \( \alpha = 2.5 \)?) In the case \( \alpha = 2.5 \) we can see that it converges faster than the case \( \alpha = 1.5 \). For \( \alpha = 0.5 \) the fixed point does not converge but seems to bounce around different values in the interval \([0, 2]\). In fact for values of \( \alpha \) between \( \frac{4}{3} \) and \( \frac{1}{2} \) we get all sorts of interesting behaviour. For more information on this click here. But when we make \( \alpha \) less than 0.5 the iteration is able to escape from the interval \([0, 2]\) and once it does this it increases rapidly with each iteration until it causes an overflow error in the computer.

Now let us see whether we can understand what is happening: First let us look at the range of the function

\[
\text{Range}(g(x)) = [0, \frac{1}{\alpha}]
\]

This shows why the iterations blow up for \( \alpha \) less than 0.5. For \( \alpha < 0.5 \) the range is not within the domain of \( g(x) \) (i.e. \([0, 2]\)) and so points may ‘escape’. However for any value of \( \alpha \) greater than \( 1/2 \) the range is mapped to within the domain.

Next we need to look at the derivative of \( g(x) \)

\[
g'(x) = \frac{2}{\alpha}(x - 1)
\]

The magnitude of the derivative is only less than 1 for all values of \( x \) if \( \alpha \geq 2 \). Thus for any value of \( \alpha \) greater than two the fixed point theorem holds and we have guaranteed convergence. We know, however, that we still get convergence to a fixed point for some values of \( \alpha \) less than two. What is happening in these cases?

---

Get matlab code used in the example from [www.physics.arizona.edu/ restrepo/475A/Notes/code1.m](http://www.physics.arizona.edu/ restrepo/475A/Notes/code1.m)
Figure 25: (a) Iterations, (b) plot of $y = g(x; \alpha = 2.5)$ and $y = x$. Here, $g(x; \alpha) = \frac{x^2 - 2x + 1}{\alpha}$ for $x \in [0, 2]$. (c) log of the error as a function of the iterate number.
Figure 26: (a) Iterations. (b) log of the error as a function of the iterate number.
Figure 27: (a) Iterations, (b) plot of $y = g(x; \alpha = 0.5)$ and $y = x$. Here, $g(x; \alpha) = \frac{x^2 - 2x + 1}{\alpha}$ for $x \in [0, 2]$. (c) log of the error as a function of the iterate number.
Figure 28: (a) Iterations, (b) plot of $y = g(x; \alpha = 0.49)$ and $y = x$. Here, $g(x; \alpha) = \frac{x^2 - 2x + 1}{\alpha}$ for $x \in [0, 2]$. (c) log of the error as a function of the iterate number.
If $\alpha < 2$ the magnitude of the derivative will be less than one for $(1 - \frac{\alpha}{2}, 1 + \frac{\alpha}{2})$. As long as the fixed point lies within this interval the theorem tells us that there will be a region around the fixed point where iterations will converge to the fixed point. This is the case as long as $\alpha > \frac{4}{3}$. As it turns out, we may still start at any point within $[0, 2]$ and we will eventually arrive at the fixed point although convergence takes longer and longer the closer $\alpha$ is to the critical point.

For values of $\alpha < \frac{4}{3}$ the fixed point still exists but it becomes unstable (i.e. If you start close to the fixed point and iterate you will move away from it rather than towards it).

If we plot $g(x)$ and the line $y = x$ on the same graph we can see that there is only one fixed point within the interval $[0, 2]$ for all values of $\alpha > 0.5$. In fact we can calculate the value of the fixed point analytically by solving $g(x) = x$.

\[
x = \frac{x^2 - 2x + 1}{\alpha} \quad \alpha x = x^2 - 2x + 1 \quad 0 = x^2 - (2 + \alpha)x + 1
\]

This is a simple quadratic equation with two solutions

\[
\hat{x} = \frac{2 + \alpha \pm \sqrt{\alpha^2 + 4\alpha}}{2}
\]

For $\alpha > 0.5$ only the smaller of the two solutions lies within the interval $[0, 2]$ and is the unique fixed point.

Remark Acceleration of Linear Methods: Is there a way to accelerate convergence of any linear method? Yes, use Aitken’s $\Delta^2$ Method (see Aitken acceleration 13.1).

5.6.1 Contraction Mapping

The fixed point theorem is the simplest version of a more general result, the contraction mapping theorem. It is very useful (and not too onerous) to find out about contraction mapping in function spaces and in certain geometries. The Brouwer contraction mapping is quite useful in nonlinear analysis, particularly because it is simple and powerful.
Figure 29: Contraction Mapping. The range of $f(x)$ between $x = a$ and $x = b$ is smaller in length than $b - a$.

**Definition:** Let $C \subseteq \mathbb{R}$ and $F : C \to C$. We say that $F$ is **contractive** if $\exists \; \lambda < 1$ such that

$$|F(x) - F(y)| \leq \lambda |x - y| \; \forall x, y \text{ in domain of } F$$

See Figure 29 for a simple contraction example.

**Contractive Mapping Theorem:** Let $C \subseteq \mathbb{R}$ closed subset. If $F : C \to C$ is contractive from $C$ to $C$, then $F$ has unique fixed point. Moreover, this fixed point is the limit of every sequence $x_{n+1} = F(x_n)$, with $x_0 \in C$ and $n = 0, 1, 2, \ldots$.

Quick Summary on Results on Fixed Point Iteration:
In what follows, let $I = [a, b]$:

- **Theorem:** If $g \in C(I)$ and $g \in I \quad \forall x \quad \in I$ then $g$ has a fixed point in $I$.

- **Theorem:** Suppose, in addition, that $g'(x)$ exists on $(a, b)$, and define $0 < k < 1$ so that $|g'(x)| \leq k < 1 \quad \forall x \in (a, b)$. Then fixed point is unique.

- **Theorem:** Let $g(I) \subseteq I \equiv [a, b]$ and $|g'(x)| \leq k < 1 \quad \forall x \in I$. For $x_0 \in I$, the sequence $x_n = g(x_{n-1})$, where $n = 1, 2 \cdots$, converges to the fixed point $s$. Furthermore, the $n^{th}$ error $e_n = x_n - s$ satisfies

$$|e_n| \leq \frac{k^n}{1-k} |x_1 - x_0|$$

**Remark:** This last result is a “nonlocal” convergence theorem because it specifies a KNOWN interval $I = [a, b]$ and gives convergence for any $x_0 \in I$. It is often the case that we don’t know the interval I, but we hope that if we pick $x_0$ sufficiently close to $s$, it would still converge: that would be a “local convergence” result.

- **Theorem:** Let $g'(x)$ be continuous in some open interval $I$ containing $s$, where $s = g(s)$. If $|g'(x)| < 1$ then there exists a $\varepsilon > 0$ such that the $x_n = g(x_{n-1})$ is convergent whenever $|x_0 - s| < \varepsilon$.

- **Corollary:** Suppose $g'(x)$ is continuous on $I$, and $g(x)$ continuous on $I$ containing $s$ and $|g'(b)| > 1$. Then there is a neighborhood of $s$ in which no initial guess (except $x_0 = s$) will work.

$\square$

### 5.7 Non-Simple Roots

What happens in the Newton and Secant Methods if $f'(p_n)$ and $f(p_n)$ go simultaneously to 0? This is a situation where we have a NON-SIMPLE root, or a root WITH MULTIPLICITY. See Figure 30 for a comparison between a simple and a multiple root. In order to deal with non-simple roots...
we need to use a variation of several techniques we’ve discussed before.

First, we define what it a *simple and non-simple* zero are: A solution \( p \) of \( f(x) = 0 \) is zero of multiplicity \( m \) if \( f(x) \) can be written as \( f(x) = (x-p)^m q(x) \) for \( x \neq p \) where \( \lim_{x \to p} q(x) \neq 0 \). Here \( q(x) \) represents the portion of \( f(x) \) not contributing to zero of \( f \).

All algorithms presented thus far have assumed the existence of a simple root. So, how do we identify whether there’s a simple root:

1) \( f \in C^1[a,b] \) has a root simple in \((a,b) \iff f(p) = 0 \text{ but } f'(p) \neq 0. \)

2) \( f \in C^m[a,b] \) has a root of \( m \) multiplicity \( p \iff 0 = f(p) = f'(p) = f''(p) \ldots f^{(m-1)}(p) \text{ but } f^m(p) \neq 0. \)

\[ \square \]

Method for handling multiple zeros:

let \( \mu = \frac{f(x)}{f'(x)} \)

if \( p \) is \( m \)- root \( m \geq 1 \) \( \Rightarrow f(x) \equiv (x-p)^mq(x) \)

\[
\mu(x) = \frac{(x-p)^mq(x)}{m(x-p)^m q(x) + (x-p)^m q'(x)} = \frac{(x-p)q(x)}{mq(x) + (x-p)q'(x)} \text{ has root } p \text{ but multiplicity 1}
\]
Now, apply Fixed Point iteration to

\[ g(x) = x - \frac{\mu(x)}{\mu'(x)} \]

\[ g(x) = x - \frac{f(x)f'(x)}{[f'(x)]^2 - f(x)f''(x)}. \]

Note that this is the difference between 2 small numbers! This could be a good source of loss of significance errors. So although we have a way to do the computation, we still have to be very careful.

\[ \square \]

5.8 Zeros of Polynomials

The problem is to find the zeros of a polynomial of the form

\[ P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \quad \text{\(n^{th}\) - degree polynomial.} \]

Theorem: Fundamental Theorem of Algebra

If \( P \) is a polynomial of degree \( n \geq 1 \), then \( P(x) = 0 \) has at least 1 (possibly complex) root.

\[ \square \]

Corollary: If \( P(x) \) is a polynomial of degree \( n \geq 1 \), then there are a unique set of constants \( x_1, x_2, \ldots x_k \), (possibly complex), and a unique set of positive integers \( m_1, m_2, \ldots m_k \), such that \( \sum_{i=1}^{k} m_i = n \) and

\[ P(x) = a_n (x - x_1)^{m_1} (x - x_2)^{m_2} \cdots (x - x_k)^{m_k} \]

Proof If a polynomial \( P \) has a root \( x_1 \), then it can be factored as \( P = (x - x_1) \ast Q \) for some polynomial \( Q \) of degree \( n - 1 \). \( Q \) in turn has a root, which can be factored. Keep going with the procedure until you find all of the roots. Some may not be unique.

\[ \square \]
Corollary Let $P$ and $Q$ be polynomials of degree at most $n$. If $x_1, x_2 \cdots x_k$ with $k > n$, are distinct numbers with $P(x_i) = Q(x_i)$ for $i = 1, 2, \ldots k$ then $P(x) = Q(x)$, $\forall x$.

Proof Consider the polynomial $R = P - Q$ and its factorization.

Suppose you want to use the Newton-Raphson method to locate approximately the zeros of $P$. It is necessary to evaluate both $P$ and its derivative at specified values. Since both $P$ and $P'$ are polynomials there's an efficient nesting procedure. The idea is to use Horner's method to evaluate the value of a polynomial and the value of its derivative.

### 5.8.1 Horner's Method

This method uses nesting, requiring $n$ multiplications and $n$ additions to evaluate any $n^{th}$ degree polynomial.

**Theorem (Horner's)**: Let $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ and $b_n = a_n$. Set $b_k = a_k + b_{k+1} x_0$ for $k = n-1, n-2, \cdots, 1, 0$, then $b_0 = P(x_0)$. Moreover, if

$$Q(x) = b_n x^{n-1} + b_{n-1} x^{n-2} \cdots b_2 x + b_1$$

then, $P(x) = (x - x_0) Q(x) + b_0$

**Proof**: Verify by direct calculation.

Example: Evaluate the polynomial

$$4x^4 + 13x^3 - x + 8$$

at $x_0 = -3$.

Here, $n = 4$. First, find the $b$'s.

$b_n = a_n = 4$, $b_k = a_k + b_{k+1} x_0$, for $k = 3, 2, 1, 0$.

Using this formula we can calculate the coefficients $b_i$ by induction:
\[
\begin{align*}
b_3 &= a_3 + b_4x_0 = 13 + 4(-3) = 1 \\
b_2 &= a_2 + b_3x_0 = 0 + 1(-3) = -3 \\
b_1 &= a_1 + b_2x_0 = -1 + -3(-3) = 8 \\
b_0 &= P(x_0) = a_0 + b_1x_0 = 8 + 8(-3) = -16
\end{align*}
\]

This gives us the following polynomials:

\[
\begin{align*}
P(x) &= (x - x_0)Q(x) + b_0 = (x + 3)Q(x) - 16 \\
Q(x) &= b_nx^{n-1} + b_{n-1}x^{n-2} + \ldots + b_2x + b_1 \\
&= b_4x^3 + b_3x^2 + b_2x + b_1 = 4x^3 + x^2 - 3x + 8 \\
P(x) &= (x + 3)(4x^3 + x^2 - 3x + 8) - 16
\end{align*}
\]

\[\square\]

\textbf{TABLE FORM:} It is now reminiscent of long division:

<table>
<thead>
<tr>
<th>Coefficient of $x_0 = -3$</th>
<th>$x^4$</th>
<th>$x^3$</th>
<th>$x^2$</th>
<th>$x^1$</th>
<th>$x^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a_4$</td>
<td>$a_3$</td>
<td>$a_2$</td>
<td>$a_1$</td>
<td>$a_0$</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>13</td>
<td>0</td>
<td>-1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>$b_4x_0$</td>
<td>$b_3x_0$</td>
<td>$b_2x_0$</td>
<td>$b_1x_0$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>-12</td>
<td>-3</td>
<td>9</td>
<td>-24</td>
<td>-16</td>
</tr>
<tr>
<td></td>
<td>$b_4$</td>
<td>$b_3$</td>
<td>$b_2$</td>
<td>$b_1$</td>
<td>$b_0$</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>-3</td>
<td>8</td>
<td>-16</td>
</tr>
</tbody>
</table>

Horner’s also yields a derivative of $P(x)$:

Since $P(x) = (x - x_0)Q(x) + b_0$ where $Q(x) = b_nx^{n-1} + b_{n-1}x^{n-2} \ldots b_2x + b_1$

differentiating, $P'(x) = Q(x) + (x - x_0)Q'(x)$,

therefore $P'(x_0) = Q(x_0)$.

\[\square\]

\textbf{ALGORITHM:}
To find \( P(x_0) \) and \( P'(x_0) \):

**Input:** \( n \), the polynomial degree, the coefficients \( a_0, a_1, \ldots, a_n \), and the value \( x_0 \).

**Output:** \( z = P(x_0) \), \( zp = P'(x_0) \).

\[
\begin{align*}
z &= a_n \\
zp &= a_n \\
\text{for } j &= n-1:-1:1 \\
& \quad z = x_0 z + a_j \\
& \quad zp = x_0 zp + z \\
\text{end} \\
\text{z} &= x_0 z + a_0 \\
\text{output } z, zp \\
\text{end}
\end{align*}
\]

**Example** Find an approximation to one of the zeros of

\[
P(x) = 4x^4 + 13x^3 - x + 8
\]

using the Newton-Raphson (see 5.3) procedure and Horner’s.

With \( x_0 = -3 \) as initial guess, we obtained previously \( P(-3) \) in previous example.

<table>
<thead>
<tr>
<th>( x_0 = -3 )</th>
<th>( a_4 )</th>
<th>( a_3 )</th>
<th>( a_2 )</th>
<th>( a_1 )</th>
<th>( a_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_4x_0 := -12 )</td>
<td>( b_3x_0 = -3 )</td>
<td>( b_2x_0 = 9 )</td>
<td>( b_1x_0 = -24 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b_4 = 4 )</td>
<td>( b_3 = 1 )</td>
<td>( b_2 = -3 )</td>
<td>( b_1 = 8 )</td>
<td>( b_0 = 16 = P(-3) )</td>
<td></td>
</tr>
</tbody>
</table>

\[
Q(x) = 4x^3 + x^2 - 3x + 8 \quad \text{and} \quad P'(-3) = Q(-3)
\]
So $P'(−3)$ can be found by evaluating $Q(−3)$:

\[
\begin{array}{cccc}
  x_0 = -3 & 4 & 1 & -3 & 8 \\
  \hline
  b'_4x_0 = -12 & b'_3x_0 = 33 & b'_2x_0 = -90 \\
  b'_4 = 4 & b'_3 = -11 & b'_2 = 30 & b'_1 = -82 = Q(−3) = P'(−3)
\end{array}
\]

Then, by Newton Raphson,

\[
x_1 = x_0 - \frac{P(x_0)}{P'(x_0)} = -3 - \frac{16}{82} \approx -3.1951.
\]

Repeating this procedure yields $x_2$:

\[
\begin{array}{cccccc}
  x_1 = -3.1951 & 4 & 13 & 0 & -1 & 8 \\
  & -12.7805 & -0.7013 & 2.2408 & -3.9646 \\
  & 4 & 0.2195 & -0.7013 & 1.2408 & 4.0354 = P(x_1) \\
  & -12.7805 & 40.1339 & -125.9921 \\
  & 4 & -12.5610 & 39.4326 & -124.7513 = Q(x_1) = P'(x_1)
\end{array}
\]

So $P(−3.1951) \approx 4.0354, P'(−3.1951) \approx −124.7513$

\[
x_2 = -3.1951 - \frac{4.0354}{-124.7513} \approx -3.1628.
\]

Keep repeating this procedure.

The root is $-3.16171$, to 5 decimal places.

\[\square\]

If the $N^{th}$-iterate $x_n$, in the Newton-Raphson procedure is the $N^{th}$ approximation of $Q$ for $P$ then

\[
P(x) = (x - x_n)Q(x) + b_0 = (x - x_n)Q(x) + P(x_n) \approx (x - x_n)Q(x)
\]

so $x - x_n$ is an approximate factor of $P$.

Let $x_n = \tilde{x}$, be the approximate zero of $P$, and $Q_1$ the approximate factored polynomial. Then

\[
P(x) \approx (x - \tilde{x})Q_1(x)
\]

103
We can find a second approximate zero of $P$ by applying Newton-Raphson to $Q_1$. If $P$ is an $n^{th}$ degree polynomial with $n$ real zeros, repeating the procedure will eventually lead to $(n - z)$ approximate zeros of $P$ and an approximate quadratic factor $Q_{n-2}(x)$. $Q_{n-2}(x) = 0$ can be solved using the quadratic formula to find the remaining 2 roots. This is called deflation, and leads to

$$P(x) = (x - \hat{x}_1)(x - \hat{x}_2) \cdots (x - \hat{x}_k)Q_k(x).$$

The difficulty with deflation is that Newton-Raphson applied repetitively compounds errors. The larger $k$, the worse $\hat{x}_{k+1}$, which is a root of $Q_k(x) = 0$, will approximate a root of $P(x) = 0$. One way to eliminate these errors is to use the reduced equations to find $\hat{x}_2, \hat{x}_3, \cdots \hat{x}_k$ approximate zeros of $P$ and then use these in a Newton Raphson procedure on $P(x)$ with these approximations as initial guesses. Also, knowing the nature of $P(x)$ and the approximate location of a root, it is easy to exploit the sign of $P(x)$ in a neighborhood of $\hat{x}_i$’s to define a good search interval for each root.

### 5.9 Müller’s Method and Complex Roots

Suppose the roots of $f(x)$ are complex. What can we do then? The algorithm still makes sense with complex values, however, if you consider what the operations in the algorithm are, you will see that real initial guesses, with polynomials with real coefficients will never yield complex roots from Newton-Raphson. Also, there is the problem of manipulating complex floating point numbers on the computer. In MATLAB this is not a problem since it takes care of complex vs. real automatically. In Fortran or C++ there is support for complex arithmetic. In c, you need to explicitly code up the complex arithmetic. In any case, you can express $P(x) = P_R(x) + iP_I(x)$ and roots $x = \alpha + i\beta$ and solve this problem as a vector equation with 2 components. However, you need to replace the standard Newton-Raphson with something like Müller’s Method, which can handle complex roots quite naturally.

Developed in (1956). Used to find zeros of arbitrary analytic functions (particularly useful in finding roots of polynomials.) Of course, the method can be used to find roots of generic functions $f(x)$, not just polynomials.
Figure 31: Finding zeros with the Secant Method. We look for the intersection of the secant and the x-axis. The approximate root is \( x_2 \).

It is a variation of the secant method (see 5.5). The process is illustrated in Figure 32. It is also useful when searching for complex roots of analytic functions. (Recall: A function is analytic at a point \( x_0 \) if the function has a convergent Taylor series at \( x_0 \). A function is analytic in a region if it is analytic at every point in the region.)

In the secant method we start with 2 initial guesses \( x_0 \) and \( x_1 \). The next approximation \( x_2 \) is the intersection with the \( x \) axis of the line \((x_0, f(x_0))\) and \((x_1, f(x_1))\). Then we continue with \( x_1 \) and \( x_2 \) etc. Müller’s method uses 3 initial guesses \( x_0, x_1, x_2 \) and determines the intersection with the \( x \) axis of a parabola, as shown in Figure 32. Note that this is done by finding the root of an explicit quadratic equation. The case where the roots are not real is handled as well, though the geometric interpretation is more complicated. (This is where the analyticity of the function is important, it makes the value of the function for a complex argument meaningful).

Consider \( P(x) = a(x - x_2)^2 + b(x - x_2) + c \)

make it pass through \((x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))\)
Figure 32: Finding zeros using Müller’s method. We locally fit a parabola and look at the intersection of the parabola with the x-axis.

Determine $a, b, c$:

\[
\begin{align*}
  f(x_0) &= a(x_0 - x_2)^2 + b(x_0 - x_2) + c \\
  f(x_1) &= a(x_1 - x_2)^2 + b(x_1 - x_2) + c \\
  f(x_2) &= a \cdot 0^2 + b \cdot 0 + c \\
  c &= f(x_2) \\
  b &= \frac{(x_0 - x_2)^2[f(x_1) - f(x_2)] - (x_1 - x_2)^2[f(x_0) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)} \\
  a &= \frac{(x_1 - x_2)[f(x_0) - f(x_2)] - (x_0 - x_2)[f(x_1) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}
\end{align*}
\]

To find $x_3$, the zero of $P$, apply the quadratic formula to $P$. There will be two roots. The root we are interested in is the one that is close to $x_2$. To avoid round-off errors due to subtraction of nearby equal numbers, use:

\[
x_3 - x_2 = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \quad 2 \text{ possibilities}.
\]

use the sign that agrees with the discriminant, i.e. the one that gives largest
denominator, and a result closer to \( x_3 \)

\[
x_3 = x_2 - \frac{2c}{b + \text{sgn}(b) \sqrt{b^2 - 4ac}}
\]

Once \( x_3 \) is determined, let \( x_0 = x_1 \quad x_1 = x_2 \quad x_2 = x_3 \) and repeat

Müller Algorithm: to find \( x \) such that \( f(x) = 0 \) with \( x_0, x_1, x_2 \), initial guesses.

input:

\( x_0, x_1, x_2 \), initial guesses
TOL error
\( N_0 \) maximum iterates

output: approximate of root or failure notification

\[
h_1 = x_1 - x_0 \\
h_2 = x_2 - x_1 \\
delta_1 = \frac{f(x_1) - f(x_0)}{h_1} \\
delta_2 = \frac{f(x_2) - f(x_1)}{h_2} \\
d = \frac{(\delta_2 - \delta_1)}{(h_2 + h_1)}
\]

\( i = 2 \)

while \( i \leq N_0 \) {

\[
b = \frac{\delta_2 + h_2 \cdot d}{\sqrt{b^2 - 4f(x_2) \cdot d}} \quad \% \text{may be complex}
\]

if \( |b - D| < |b + d| \) then

\[
E = b + D
\]
else

\[
E = b - D
\]
end

\[
h = -\frac{2f(x_2)}{E}
\]

\[
p = x_2 + h
\]

if \( |h| < \text{TOL} \) then

output \( p \) \% an approximation to the root
STOP
end

\( x_0 = x_1 \)
\( x_1 = x_2 \)
Consider the function \( f(x) = e^x + 1 \). Clearly this has no real roots. Suppose that we start with three initial “guesses” \( x_0 = 1 \) and \( x_1 = 0 \) and \( x_2 = -1 \), Müllers method would then lead us to find the quadratic polynomial passing through \((1,e+1),(0,2)\) and \((-1,1+1/e)\). That is \( P(x) = a(x+1)^2 + b(x+1) + c \)

\[
\begin{align*}
\text{\( c \)} &= f(x_2) = 1 + 1/e \approx 1.3679 \\
\text{\( b \)} &= \frac{(x_0 - x_2)[f(x_1) - f(x_2)] - (x_1 - x_2)[f(x_0) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)} \\
&= \frac{4(1 - 1/e) - (e - 1/e)}{2} \approx 0.0890 \\
\text{\( a \)} &= \frac{(x_1 - x_2)[f(x_0) - f(x_2)] - (x_0 - x_2)[f(x_1) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)} \\
&= \frac{(e - 1/e) - 2(1 - 1/e)}{2} \approx 0.5431
\end{align*}
\]

\( x_3 \) is found as the root of \( P(x) \) that is close to \( x_2 \), that is

\[
x_3 - x_2 = x_3 + 1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \approx -0.820 + 1.5849i
\]

so \( x_3 \approx -1.0820 + 1.5849i \) we use the positive sign in front of the square root in the denominator to match the sign of \( b \) in order to choose the value of \( x_3 - x_2 \) with smallest absolute value. Of course, in this case since the term under the square root is negative, the two roots have the same absolute value, but, we choose this one even so, since that is the way the algorithm is
defined. However, this raises the issue of how to pick the sign of the square root when $b$ may not be real. The guiding principal is to always make the choice that picks the root of the quadratic that is closest to our most recent estimate. At the next iteration, we get

$$a \approx 0.2157 + 0.1094i \quad b \approx 0.0343 + 0.5563i \quad c \approx 0.9952 + 0.3389i$$

the formula for $x_4$ is $x_4 - x_3 = \frac{-2c/b \pm \sqrt{b^2 - 4ac}}{b}$. The two possible values of the denominator are approximately $0.3595 - 0.5040i$ and $-0.2909 + 1.6167i$, the larger choice is the latter thus we use the negative sign in defining $x_4$. This sort of checking is exactly what is in the algorithm listed above. We could avoid the comparison by rewriting:

$$x_4 - x_3 = \frac{-2cb^*/(|b|^2 + \sqrt{|b|^4 - 4ac(b^*)^2})}$$

where $b^*$ is the complex conjugate of $b$, and the square root is always chosen so the result has non-negative real part. In this example, if we continue, the algorithm will converge to the root $x = \pi i$. Here is a table of the next few values:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0000</td>
</tr>
<tr>
<td>1</td>
<td>0.0000</td>
</tr>
<tr>
<td>2</td>
<td>-1.0000</td>
</tr>
<tr>
<td>3</td>
<td>-1.0820 + 1.5849i</td>
</tr>
<tr>
<td>4</td>
<td>-1.2735 + 2.8505i</td>
</tr>
<tr>
<td>5</td>
<td>-0.3814 + 3.7475i</td>
</tr>
<tr>
<td>6</td>
<td>0.1973 + 3.1457i</td>
</tr>
<tr>
<td>7</td>
<td>-0.0168 + 3.1571i</td>
</tr>
<tr>
<td>8</td>
<td>0.0000 + 3.1421i</td>
</tr>
<tr>
<td>9</td>
<td>0.0000 + 3.1416i</td>
</tr>
</tbody>
</table>

6 \hspace{1cm} NORMS OF FUNCTIONS

$$L_1 - \text{norm} \quad \|f - q\|_1 = \int_a^b |f(x) - q(x)|w(x)dx$$
\[ L_2 - \text{norm} \quad \| f - q \|_2 = \left\{ \int_a^b |f(x) - q(x)|^2 w(x) \, dx \right\}^{\frac{1}{2}} \]

\[ L_p - \text{norm} \quad \| f - q \|_p = \left\{ \int_a^b |f(x) - q(x)|^p w(x) \, dx \right\}^{\frac{1}{p}} \]

\[ L_\infty - \text{norm} \quad \| f - q \|_\infty = \max_{a \leq x \leq b} |f(x) - q(x)| \]

\( w(x) \) is a “weight" function, which provides some flexibility in measuring closeness. In all cases consider \( w(x) \) is continuous and non-negative on \((a, b)\).

\( \int_a^b w(x) \, dx \) exists and \( \int_a^b w(x) \, dx > 0 \).

**Remarks:** A sequence of functions \( \{g_k(x)\}_{k=1}^\infty \) is said to converge to \( g(x) \) with regard to a given norm \( \| \cdot \| \) if

\[ \lim_{k \to \infty} \| g_k - g \| = 0 \]

Advanced calculus students recognize \( \| \cdot \|_\infty \)-norm is equivalent to uniform convergence.)

In scientific computing we obsess over quantifying things. The point of this example is to show that there are many ways to quantify how close one function is from another one, and that thought should be placed on what is the most useful way to quantify this closeness.

**Example** In this example we want to compare quantitatively a function to another one. We might want to do this in order to quantify how well one of these functions approximates the other one.\(^6\)

Take \( x \in [a, b] \quad a = 0, b = 3 \) in what follows:

Let \( \theta(x) \equiv 0 \) (zero function) \quad \( x \in [a, b] \). Consider this the function we want to approximate. We are going to propose a 1-parameter family of functions \( f_k(x) \) as candidate approximating functions to \( \theta(x) \). These are defined as follows:

\(^6\)I learned this example in my own class-taking.
Figure 33: Approximating $\theta(x)$ with $f_k(x)$.

For any $k > 0$ let $f_k(x)$ be given by

$$
\begin{align*}
  f_k(x) = \begin{cases}
    k(k^2x - 1) & \frac{1}{k^2} \leq x \leq \frac{2}{k^2} \\
    -k(k^2x - 1) & \frac{2}{k^2} \leq x \leq \frac{3}{k^2} \\
    0 & \text{otherwise}
  \end{cases}
\end{align*}
$$

here $k \geq 1$. See Figure 33 for an illustration.

We obtain the following from these norms:

\begin{align*}
  ||\theta - f_k||_1 &= \frac{1}{k} \\
  ||\theta - f_k||_2 &= \frac{\sqrt{2}}{\sqrt{3}} \\
  ||\theta - f_k||_\infty &= k.
\end{align*}

So, the $L_1$-norm drops as $k$ increases, the $L_2$-norm stays constant, the $L_\infty$-norm grows as $k$ increases. Which norm should we use?
Now, consider $\{||f - f_k||\}_{k=1}^\infty$ a sequence of real numbers and see
\[
\lim_{k \to \infty} ||\theta - f_k||_1 = 0
\]
\[
\lim_{k \to \infty} ||\theta - f_k||_2 = \sqrt{2}/3
\]
\[
\lim_{k \to \infty} ||\theta - f_k||_\infty = \infty
\]
therefore the sequence $\{f_k(x)\}_{k=1}^\infty$ converges to $\theta(x)$ ONLY in the $||\cdot||_1$, norm.

In many practical cases we would probably accept $f_k(x)$ for large $k$ as a “good” approximation for $f = \theta(x)$ since it is only “bad” in a small neighborhood of a single point. However, there are practical problems in which even moderate errors in a small neighborhood are unacceptable.

Useful facts about $L_1$, $L_2$, and $L_\infty$:

Let $w(x) := 1$, $b \geq a$.

\* \[
||f - q||_1 = \int_a^b |f(x) - g(x)|w(x)dx \leq \max_{a \leq x \leq b} |f(x) - g(x)| \int_a^b w(x)dx
\]
\[
= ||f - q||_a \left( \int_a^b w(x)dx \right)
\]
\[
\text{or } ||\cdot||_1 \leq ||\cdot||_\infty \int_a^b w(x)dx \text{ constant}
\]

\* \[
||f - q||_2 \leq ||f - q||_\infty \left( \int_a^b wdx \right)^{1/2}
\]
\[
\text{or } ||\cdot||_2 \leq ||\cdot||_\infty \left( \int_a^b w(x)dx \right)^{1/2}
\]

\* As a consequence, if $||\cdot||_\infty$ small then both $||\cdot||_1$ and $||\cdot||_2$ are smaller. Thus $||\cdot||_\infty$ is stronger than the other 2 norms. Usually, we strive for this in our work.
7 INTERPOLATION

7.1 Preliminaries

Problem: suppose we are given $n+1$ paired real values $x$ and $y$. For example, they are given to us as a table of data:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$\cdots$</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>$y_0$</td>
<td>$y_1$</td>
<td>$\cdots$</td>
<td>$y_n$</td>
</tr>
</tbody>
</table>

$n+1$ values

Our problem is to find a polynomial $p$ of lowest degree for which

$$p(x_i) = y_i \quad 0 \leq i \leq n.$$  

Here, $p$ is the interpolating polynomial of the data.

**Theorem 1:** If $x_0, x_1, \cdots, x_n$ are distinct real numbers then, for arbitrary values $y_0, y_1, \cdots, y_n$ there exists a unique polynomial $p_n$ of degree at most $n$ such that

$$p_n(x_i) = y_i, \quad 0 \leq i \leq n.$$  

where $p_n(x) = a_0 + a_1 x^1 + \cdots + a_n x^n$ with $n \geq 0$.

**Remark:** We will be primarily concerned with real, continuous functions over some closed interval $[a, b]$. For simplicity, we are assuming that $f(x)$ is real and $f(x) \in C[a, b]$.

Why interpolate with polynomials? They approximate continuous functions uniformly. That is, given any function, defined and continuous on a closed interval, there exists a polynomial that is as close to the given function as desired in the infinity norm (the $L_\infty$-norm):
Weierstrass Approximation Theorem: If $f$ is defined and continuous on $[a, b]$ and $\varepsilon > 0$ is given, then there exists polynomials $p_\varepsilon$, defined on $[a, b]$, with the property

$$|f(x) - p_\varepsilon| < \varepsilon \quad \forall x \in [a, b].$$

The situation is depicted in Figure 34. This theorem extends to $f \in C^0[a, b]$ functions.

\[ \tag*{\Box} \]

Remark: $g(x) = f(a) + f'(a)(x - a) + \cdots + \frac{f^{(k)}(x - a)^k}{k!}$ is a $k^{th}$ degree polynomial that approximates $f(x)$ for $x$ near $a$. If $f^{(k+1)}(x)$ is continuous then the error at $x$ is given by

$$\frac{f^{(k+1)}(\xi)}{(k + 1)!}(x - a)^{k+1} \quad \xi \text{ between } x \text{ and } a.$$  

\[ \tag*{\Box} \]

Remark: Suppose $g(x)$ approximates $f(x)$ and we wish to find a numerical
approximation for
\[ \int_a^b f(x)dx \text{ or } f'(\alpha) \text{ for some } x = \alpha. \]

Then we would use
\[ \int_a^b q(x)dx \text{ or } q'(\alpha). \]

However it may be that while
\[ \int_a^b q(x)dx \approx \int_a^b f(x)dx \]
it may not be true that \( q'(\alpha) \sim f'(x) \) and vice versa.

Thus, knowledge of how these approximations are subsequently going to be used determines the manner in which \( f(x) \) should be approximated by some simpler function.

\[ \square \]

How do we measure how close two functions \( f(x) \) and \( g(x) \) are? We want to quantify \( f(x) - g(x) \) as “small” in some sense, if \( f(x) \approx g(x) \):

**Theorem** Set \( f(x) \in C[a, b] \) and let \( n > 0 \) be an integer. If \( \| \cdot \| \) is any one of the norms \( L_1, L_2, \) or \( L_\infty \) (discussed previously) then there exists a unique polynomial \( p^*(x) \) of degree \( n \) or less such that \( \| f - p^*(x) \| \leq \| f - p \| \) for all \( p(x) \in p_n(x) \)

\[ \square \]

**Remark** This theorem tells us that there exists a unique best \( n^{th} \)-degree polynomial approximation to \( f(x) \) in the \( L_p \) norm \( (p = 1, 2, \infty) \). The preceding theorem might seem completely obvious. You can have sets where there is no minimal element. In the theorem, if we omit the condition on the degree of the polynomial, i.e. on \( n \), and consider all polynomials, then there is no best polynomial approximant.

### 7.2 Polynomial Interpolation

The problem is to interpolate \( f(x) \) by \( p(x) \in \mathcal{P}_n \) in \( C[a, b] \).
Definition $\mathcal{P}_n$ is defined as the set of all polynomials of degree $n$ or less. For example,

$$\mathcal{P}_3 = \{ax^3 + bx^2 + cx + d | a, b, c, d, \text{real} \}.$$ 

This is equivalent to saying the linear algebraic statement that $\mathcal{P}_n$ is defined as a linear combination of functions from the set $\{x^0, x^1, x^2, \ldots, x^n\}$.

Let $x_0, x_1, \ldots, x_n$ be $n + 1$ distinct points on interval $[a, b]$. Then $p(x) \in \mathcal{P}_n$ is said to interpolate $f(x)$ at each of these points if

$$p(x_j) = f(x_j) \quad 0 \leq j \leq n.$$ 

Theorem (Existence and Uniqueness)

Let $\{x_j\}^{n}_{j=0}$ be $(n + 1)$ distinct points in $[a, b]$. Let $\{y_j\}^{n}_{j=0}$ be any set of real numbers, then there exists a unique $p(x) \in \mathcal{P}_n$ such that $p(x_j) = y_j$ for $j \in [0, n]$.

Proof: (Existence) For each $j$, $0 \leq j \leq n$, let $\ell_j(x)$ be the $n^{th}$ degree polynomial defined by

$$\ell_j(x) \equiv \frac{(x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_0)(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)}$$

$$\equiv \prod_{\substack{i=0 \atop i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}$$

These are called **Cardinal Functions**.

Definition The Kronecker Delta Function $\delta_{ij}$ is defined as

$$\delta_{ij} = \begin{cases} 
1 & i = j \\
0 & i \neq j \end{cases}$$

Remark $\ell_i(x_j) = \delta_{ij}$

Since the sum of the $n^{th}$ degree polynomials is again a polynomial of at most $n^{th}$ degree, then let

$$p(x) \equiv \sum_{j=0}^{n} y_j \ell_j(x) \quad \text{“Lagrange Form of the interpolating polynomials”}$$

116
Figure 35: Illustration of $\ell_j(x)$, for $x \in [-1, 1]$, for (a) $n$ even, (b) $n$ odd.

For $0 \leq i \leq n$

$$p(x_i) = y_0\ell_0(x_i) + y_1\ell_1(x_i) + \cdots + y_n\ell_n(x_i) = y_i\ell_i(x_i) = y_i$$

A sketch of $\ell_j(x)$ when $n$ is even and $n$ odd appears in Figure 35, for $x_i$ equidistant.

If $f(x)$ is a function such that $f(x_i) = y_i$ $0 \leq i \leq n$, then there is an interpolating polynomial of the form

$$p(x) = \sum_{j=0}^{n} f(x_j)\ell_j(x)$$

that belongs to $P_n$.

(Uniqueness): assume there are two different polynomials $p(x) \in P_n$, $q(x) \in P_n$ such that $p(x_j) = q(x_j) = y_i$ for $0 \leq j \leq n$. If $r(x) = p(x) - q(x)$ then $r(x) \in P_n$ and $r(x_j) = p(x_j) - q(x_j) = 0$ for $0 \leq j \leq n$. By the Fundamental Theorem of Algebra $r(x) \equiv 0$, so $p = q$, which is a contradiction.

\[\square\]

The following proof is presented because it introduces the “Method of undetermined coefficients”. The method’s name applies to a number of situations in which a number of unknown constants need to be determined, say $M$ of them. This is done by using the context of the problem to generate $M$ linearly independent equations which can uniquely determine the constants. It
will show up in integration problems (see 9), and the ODE section of the notes for 475B.

**Proof 2:** Let \( p(x) = a_0 + a_1 x + \cdots + a_n x^n \), where \( a_j \)'s are to be determined. Consider the \((n+1)\) equations

\[
p(x_j) = a_0 + a_1 x_j + a_2 x_j^2 + \cdots + a_n x_j^n \quad 0 \leq j \leq n
\]

In matrix form

\[
\begin{bmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^n \\
1 & x_1 & x_1^2 & \cdots & x_1^n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_n & x_n^2 & \cdots & x_n^n \\
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_n \\
\end{bmatrix}
= \begin{bmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_n \\
\end{bmatrix}
\]

(15)

or

\[
V a = y,
\]

(16)

The matrix \( V \) is called a “Vandermonde” Matrix and it is non-singular if the \( x_0, x_1, \cdots, x_n \) are distinct: \( V \) is nonsingular if and only if \( 0 \) is the only solution of \( V a = 0 \) (or determinant of \( V \neq 0 \)).

So if \( a \) is any solution of \( V a = 0 \) then \( p(x) = a_0 + a_1 x + \cdots + a_n x^n \) is an \( n^{th} \) degree polynomial with \( p(x_j) = 0, \quad j = 0, 1, \ldots, n \). Since the only \( n^{th} \) degree polynomial with \( (n+1) \) zeros is the zero polynomial, it must be that \( a = 0 \) therefore \( V \) is non-singular. Thus (15) must have a unique solution and the polynomial \( p(x) \) is found by solving (16), and is the unique interpolating polynomial in \( P_n \).

**Remarks**

1. if \( f(x) \in P_n \) then \( f(x) = p(x) \forall x \), by the uniqueness theorem.

2. The solution of the Vandermonde matrix problem, (16), may yield \( a_n = 0 \) (other coefficients may also be zero), so \( p(x) \) may be a polynomial of degree strictly less than \( n \).

3. if \( m > n \) there are an infinite number polynomials \( q(x) \in P_m \) which satisfy \( q(x_j) = y_j \ 0 \leq j \leq n \).
Example Suppose we want a second degree polynomial \( p(x) \) such that

\[
\begin{array}{c|ccc}
x & 0 & 1 & 2 \\
p & -1 & 2 & 7
\end{array}
\]

Writing the required polynomials explicitly:

\[
\ell_0(x) = \frac{(x - 1)(x - 2)}{2} \quad \ell_1(x) = -x(x - 2) \quad \ell_2(x) = \frac{x(x - 1)}{2}
\]

Therefore \( p(x) = -\ell_0(x) + 2\ell_1(x) + 7\ell_2(x) = x^2 + 2x - 1 \)

Using the Method of Undetermined coefficients: since \( x_0 = 0, \ x_1 = 1, \ x_2 = 2 \):

Let

\[
p(x_j) = a_0 + a_1 x + a_2 x^2
\]

\[
p(0) = a_0 = -1 \quad p(1) = a_0 + a_1 + a_2 = 2 \quad p(2) = a_0 + 2a_1 + 4a_2 = 7.
\]

\[
\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 7 \end{bmatrix}
\]

Solving gives \( p(x) = x^2 + 2x - 1 \)

\[\square\]

Remark Often the \( y_j \)'s are determined by some function \( f(x) \). In this case the equations are \( p(x_j) = f(x_j), \ 0 \leq j \leq n \).

Example Use nodes \( x_0 = 2, \ x_1 = 2.5, \ x_2 = 4 \) to find the 2nd degree interpolating polynomial for \( f(x) = 1/x \).

\[
\ell_0(x) = \frac{(x - 2.5)(x - 4)}{(2 - 2.5)(2 - 4)} = (x - 6.5)x + 10
\]

\[
\ell_1(x) = \frac{(x - 2)(x - 4)}{(2.5 - 2)(2.5 - 4)} = \frac{(-4x + 24)x - 32}{3}
\]

\[
\ell_2(x) = \frac{(x - 2)(x - 2.5)}{(4 - 2)(4 - 2.5)} = \frac{(x - 4.5)x + 5}{3}
\]

119
Figure 36: The function \( f(x) = 1/x \) and its 2\(^{nd} \) degree interpolation \( p(x) = (0.05x - 0.425)x + 1.15 \).

Since \( f(2) = 0.5, f(2.5) = 0.4, f(4) = 0.25 \) then \( p(x) = \sum_{j=0}^{2} f(x_j)\ell_j(x) = 0.05x^2 - 0.425x + 1.15 \) The comparison between the interpolation and the function appear in Figure 36

**Theorem:** If \( x_0, x_1, \ldots, x_n \) are distinct numbers and \( f \in C^{n+1}[a,b] \) then, for each \( x \in [a,b] \) a number \( \xi(x) \) in \( (a,b) \) exists with

\[
 f(x) = p(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n)
\]

where \( p(x) = \sum_{j=0}^{n} f(x_j)\ell_j(x) \).

This error formula is important because the \( \ell_j(x) \) are used extensively for deriving numerical differentiation and integration methods. Error bounds for these are obtained from the Lagrange error formula, that follows next.
The Error in a Polynomial Interpolation

**Theorem** Let $f \in C^{n+1}[a,b]$ and $p$ be the polynomial of at most degree $n$ that interpolates $f$ at $n+1$ points $x_0, x_1, \ldots, x_n \in [a, b]$. For each $x \in [a,b]$ there corresponds a point $\xi \in (a,b)$ such that

$$f(x) - p(x) = \frac{1}{(n + 1)!} f^{(n+1)}(\xi(x)) \prod_{i=0}^{n} (x - x_i), \quad (18)$$

for some $\xi(x)$ in $[a,b]$.

**Proof:** Let $R \equiv f - p(x)$. Since (18) is trivially satisfied at $x_0, x_1, \ldots, x_n$, we need only to focus on the case where $x$ is not one of these locations.

Let $Q \equiv \prod_{i=0}^{n} (x - x_i)$ and keeping $x$ fixed, consider $g : [a,b] \rightarrow \mathbb{R}$, the whole real line, with

$$g(z) = f(z) - p(z) - Q(z) \frac{f(x) - p(x)}{Q(x)}$$

and $z \in [a,b]$. By the continuity assumption of $f$, the function $Q$ is $n + 1$-times continuously differentiable, and by construction, has $n + 1$ zeros, at the locations $x_0, x_1, \ldots, x_n$. Then, by Rolle’s Theorem, the derivative $Q'$ must have at least $n$ zeros. Repeating the argument, by induction, we deduce that the derivative $Q^{(n)}$ has at least one zero in $[a,b]$, which we denote $\xi$. For this zero we have that

$$0 = f^{(n+1)}(\xi) - (n + 1)! \frac{R}{Q},$$

and from this we obtain (18).

**Example:** We wish to approximate the $\sin(x)$ function on the interval $[-\pi, \pi]$. The chosen interpolating points are $x_0 = -\frac{2\pi}{3}$, $x_1 = \frac{-\pi}{3}$, $x_2 = \frac{\pi}{3}$ and $x_3 = \frac{2\pi}{3}$. Note that the points are unevenly spaced.

We know that

$$\sin(x_0) = \sin(x_1) = \frac{-\sqrt{3}}{2}$$

and

$$\sin(x_2) = \sin(x_3) = \frac{\sqrt{3}}{2}.$$
The Cardinal functions are as follows

\[
\begin{align*}
l_0(x) &= \frac{(x + \frac{\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3})}{(x_0 + \frac{\pi}{3})(x_0 - \frac{\pi}{3})(x_0 - \frac{2\pi}{3})} \\
&= \frac{-9(x + \frac{\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3})}{4\pi^3} \\
l_1(x) &= \frac{(x + 2\frac{\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3})}{(x_1 + 2\frac{\pi}{3})(x_1 - \frac{\pi}{3})(x_1 - \frac{2\pi}{3})} \\
&= \frac{9(x + 2\frac{\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3})}{2\pi^3} \\
l_2(x) &= \frac{(x + 2\frac{\pi}{3})(x + \frac{\pi}{3})(x - \frac{\pi}{3})}{(x_2 + 2\frac{\pi}{3})(x_2 + \frac{\pi}{3})(x_2 - \frac{2\pi}{3})} \\
&= \frac{-9(x + 2\frac{\pi}{3})(x + \frac{\pi}{3})(x - \frac{2\pi}{3})}{2\pi^3} \\
l_3(x) &= \frac{(x + 2\frac{\pi}{3})(x + \frac{\pi}{3})(x - \frac{\pi}{3})}{(x_3 + 2\frac{\pi}{3})(x_3 + \frac{\pi}{3})(x_3 - \frac{\pi}{3})} \\
&= \frac{9(x + 2\frac{\pi}{3})(x + \frac{\pi}{3})(x - \frac{\pi}{3})}{4\pi^3}.
\end{align*}
\]

So the interpolating function, \( p(x) \) is equal to

\[
\begin{align*}
p(x) &= -\frac{\sqrt{3}}{2}l_0(x) - \frac{\sqrt{3}}{2}l_1(x) + \frac{\sqrt{3}}{2}l_2(x) + \frac{\sqrt{3}}{2}l_3(x) \\
&= \frac{9\sqrt{3}}{8\pi^3} \left( (x + \frac{\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3}) - 2(x + \frac{2\pi}{3})(x - \frac{\pi}{3})(x - \frac{2\pi}{3}) \\
&\quad - 2(x + \frac{2\pi}{3})(x + \frac{\pi}{3})(x - \frac{3\pi}{3}) + (x + \frac{2\pi}{3})(x + \frac{\pi}{3})(x - \frac{3\pi}{3}) \right)
\end{align*}
\]

Figure 37 shows the interpolating function as well as \( \sin(x) \) the function which we are approximating. Equation (18) tells us that the error for this function is equal to

\[
\begin{align*}f(x) - p(x) &= \frac{1}{4!}f^{(4)}(\xi(x)) \prod_{i=0}^{3}(x - x_i) \\
&= \frac{1}{24}\sin(\xi(x)) \prod_{i=0}^{3}(x - x_i),\end{align*}
\]
Figure 37: Plot of function $\sin(x)$ (solid line) and its interpolating polynomial $p(x)$ (dashed line) for the points $x_0 = -\frac{2\pi}{3}$, $x_1 = -\frac{\pi}{3}$, $x_2 = \frac{\pi}{3}$ and $x_3 = \frac{2\pi}{3}$. The dots show the interpolation points of the polynomial approximation.
Figure 38: Plot of the error |sin(x) − p(x)| (solid line) and the estimate for the error, $\frac{1}{24} \prod_{i=0}^{n} (x - x_i)$ (dashed line). Note that the only points where the estimate of the error agrees with the error is at the interpolation points where we know it is zero.

where $\xi(x) \in [-\frac{2\pi}{3}, \frac{2\pi}{3}]$.

The maximum absolute value of the function $\sin(x)$ is 1, so we know that the absolute error will be less than $\frac{1}{24} \prod_{i=0}^{n} (x - x_i)$. Figure 38 shows the actual error of the interpolation and this approximation. As you can see the error is usually considerably smaller than this estimate.

We end with an algorithm called “Neville’s Recursive Algorithm” for the generation of the polynomials:

Input: (x0, x1, x2, ....xn)
and f(x0), f(x1), f(x2), ...f(xn),
as the first column S(0,0) S(1,0), S(2,0), ..., S(n,0) of the matrix S.
Output: the matrix $S$, with $p(x) = S(n,n)$.

for $i = 1$: $n$
    for $j = 1$: $i$
        
        $S(i,j) =
        \left[(x - x(i-j)) S(i,j-1) - (x - xi)S(i-1,j-1)\right]/[xi - x(i - j)]$
    end
end

Output $(S)$;
STOP

(you can get the code from [www.physics.arizona.edu/~restrepo/475A/Notes/neville.m](http://www.physics.arizona.edu/~restrepo/475A/Notes/neville.m).
Note that this code generates the function value corresponding to the Lagrange polynomial).

### 7.3 Chebyshev Polynomials

Chebyshev Polynomials are orthogonal polynomials, defined on the interval $x \in [-1,1]$. They are easily generated via a *recursion relation*. They are orthogonal with respect to a weight

$$w(x) = \frac{1}{\sqrt{1-x^2}}$$

One very common use for them is to choose interpolation points judiciously in order to minimize the error generally incurred in polynomial interpolation. The idea here is to choose the interpolation points so that the error term in (18) is reduced as much as possible.

Chebyshev polynomials obey the *recurrence relations*

\[
\begin{align*}
T_0(x) &= 1 & T_1(x) &= x \\
T_{n+1}(x) &= 2xT_n(x) - T_{(n-1)}(x) & n \geq 1
\end{align*}
\]

Here are first six polynomials:

\[
\begin{align*}
T_0(x) &= 1 \\
T_1(x) &= x
\end{align*}
\]
The first four Chebyshev polynomials are illustrated in Figure 39.

\[
\begin{align*}
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x
\end{align*}
\]

Theorem: For \( x \in [-1, 1] \), the Chebyshev polynomials can be expressed as

\[
T_n = \cos(n \cos^{-1}(x)) \quad n \geq 0
\]

and obey the following Properties:

1. \(|T_n(x)| \leq 1 \quad [-1, -1]
2. \(T_n(1) = 1\)
3. \(T_n(-1) = T_{2n}(0) = (-1)^n\)
4. \(T_n(\cos(\theta)) = 0, \quad \theta = \frac{(2k-1)\pi}{2n}, \quad k = 1, 2, ..., n\)
5. \(T_n(\cos(\theta)) = (-1)^k, \quad \theta = \frac{kn}{n}, \quad k = 0, 1, 2, ..., n\)
6. \(T_{2n+1}(0) = 0\)
7. There are \( n \) roots and these are distinct.

8. Polynomials, defined as \( \frac{T_n(x)}{2^{n-1}} \), have a leading coefficient of 1 (the name given to polynomials which have a coefficient 1 in the highest power are known as monic polynomials).

9. The bound on the \( n^{th} \)-degree Chebyshev polynomial, in the monic form, is given by:

\[
\max_{-1 \leq x \leq 1} \left| \frac{1}{2^{n-1}} T_n(x) \right| = \frac{1}{2^{n-1}}
\]

It might seem that a polynomial that is only defined on a specific interval, in this case for \( x \in [-1, 1] \), would have limited utility. This is not the case, since any bounded interval can be mapped to this interval and vice versa:

**Translating the Interval**

The polynomials, although defined on the interval \([-1, 1]\), can be used to interpolate data and approximate functions on arbitrary finite intervals: The domain of \( f(x) \), say, and \( p(x) \), say, from \([a, b] \rightarrow [-1, 1]\). More generally, suppose we want to compute on \([c, d]\) but need to translate to \([a, b]\) if \( t \in [c, d] \) we can perform a straight line transformation into \( x \in [a, b] \)

\[
x = \left( \frac{b - a}{d - c} \right) t + \left( \frac{ad - bc}{d - c} \right)
\]

**Optimizing Interpolation by Polynomials:**

Suppose we are interpolating a sufficiently smooth function \( f(x) \) (scaled to the interval \([-1, 1]\), as prescribed above). If we have some flexibility over the choice of interpolating knots \( x_i \) and the number of these, then we can optimize the quality of the interpolation by minimizing the error.

We are going to use Lagrange polynomials, but we’ll choose the knots on which we will interpolate: \(-1 < x_0 < x_1 \ldots < x_n < 1\). For arbitrary and
distinct interpolation knots we obtain the sup-norm error

\[ ||f(x) - p(x)||_{\infty} \leq \frac{1}{(n + 1)!} \max_{|x| \leq 1} f^{(n+1)}(\xi(x)) \max_{|x| \leq 1} \left| \prod_{i=0}^{n} (x - x_i) \right| . \]  

in the interpolation using Lagrange Polynomials.

We now ask whether this error can be minimized. The only place there is room for improvement is in the choice of the points \( x_i \). That is, we could choose \( x_i \) such that

\[ \max_{|x| \leq 1} \left| \prod_{i=0}^{n} (x - x_i) \right| \text{ is minimized}, \]

where \( \prod_{i=0}^{n} (x - x_i) \) is an \((n + 1)^{th}\) order polynomial.

This is in fact possible, as we shall see next:

Let \( x_i \) be the zeros of the \( \frac{1}{2\pi} T_{n+1}(x) \) polynomial. These are,

\[ x_i = \cos \left( \frac{(2j + 1) \pi}{2n} \right) \quad 0 \leq j \leq n. \]

(Note, by the way that

\[ x_j = \cos \frac{j \pi}{n + 1}, \quad j = 0, 1, 2, \ldots, n + 1 \]

gives the extremas).

With this choice of knots, (20) becomes

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

Statement (21) can be proven as follows: Assume there exists a monic polynomial \( V(x) \in \mathcal{P}_{n+1} \) and \( ||V||_{\infty} < || \prod_{i=0}^{n} (x - x_i) ||_{\infty} \).
If \( i \) is even, then \( V \left( \cos \left( \frac{i \pi}{n+1} \right) \right) < \prod_{i=0}^{n} (x - \cos \left( \frac{i \pi}{n+1} \right)) = \frac{1}{2^n} \) for \( 0 \leq i \leq n+1 \). Or else, \( \|V\|_{\infty} > \|\Pi(x-x_i)\|_{\infty} \).

If \( i \) is odd then \( V \left( \cos \frac{i \pi}{n+1} \right) > \prod_{i=0}^{n} \left( x - \cos \left( \frac{i \pi}{n+1} \right) \right) = -\frac{1}{2^n} \), or again,

\[
\|V\|_{\infty} \geq \|\Pi(x-\cos \frac{i \pi}{n+1})\|_{\infty}.
\]

Thus \( V(x_0) < W(x_0), V(x_1) > W(x_1), V(x_2) < W(x_2), V(x_3) > W(x_3) \), etc. Where \( W = \Pi \left( x - \cos \frac{i \pi}{n+1} \right) \).

Let \( H(x) \equiv V(x) - W(x) \), which has a zero in each interval \( (x_i, x_{i+1}) \) \( 0 \leq i \leq n \). Moreover \( H(x) \in \mathcal{P}_n \) and is monic. Thus, \( H(x) \in \mathcal{P}_n \) has at least \( (n+1) \) zeros, but \( H(x) \) has at most \( n \) zeros, therefore we conclude that such \( V(x) \) does not exist.

Note: One can also prove that

\[
\max_{|x| \leq 1} \left| \frac{1}{2^n} T_{n+1}(x) \right| = \frac{1}{2^n} \leq \max_{|x| \leq 1} |Q(x)|
\]

where \( Q(x) \) is any monic polynomial of degree \( n+1 \) with \( n+1 \) roots in \(-1 \leq x \leq 1\).

Optimizing the error in polynomial interpolation goes well beyond just the interpolation process: since \( \ell_j(x) \) are used extensively in deriving numerical differentiation and integration schemes the choice of interpolation points can be used to minimize errors in integration and differentiation and the estimates related to errors in these two operations rely on the error estimate just discussed.

\[\square\]

### 7.4 Newton’s Divided Differences and Interpolation

An efficient way to derive interpolation formulas (and thus differentiation and integration formulas) is via \textit{divided differences}. 

129
The problem is that expressing a polynomial as
\[ p(x) = y_0l_0(x) + y_1l_1(x) + \ldots + y_nl_n(x) \]
is cumbersome and somewhat inefficient. A more convenient structure is
\[ p(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) + \ldots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}), \]
which is known as the Newton form. Here the \( a_i, i = 1..n \), are constants.

Recall that the interpolation problem is based upon the following constraints:
\[ p(x_0) = f(x_0), p(x_1) = f(x_1), \ldots, p(x_n) = f(x_n), \]
where \( p \) is the interpolating polynomial and \( f \) is the function being interpolated.

So the convenient aspect of the Newton form is that the \( a_i \) can be found quickly: The \( a_i \)'s in (22) are found using (23). the first two,
\[ p(x_0) = f(x_0) \]
\[ p(x_1) = f(x_0) + a_1(x_1 - x_0) = f(x_1) \]
Therefore
\[ a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}. \]
Once you have \( a_1 \) it is easy to derive \( a_2 \) algebraically. In other words, you avoid having to solve a system of equations, and coding-wise, the procedure is recursive and thus very compact computationally.

Let
\[ f[x_i] \equiv f(x_i). \]
We define the first divided difference to be
\[ f[x_i, x_{i+1}] \equiv \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}. \]
After \( k - 1 \) divided differences we obtain
\[ f[x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k-1} \text{ and } f[x_{i+1}, x_{i+2}, \ldots, x_{i+k-1}, x_{i+k}]. \]
Hence, the \( k^{th} \) divided difference satisfies

\[
f[x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k-1}, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}
\]

and the \( a_k = f[x_0, x_1, x_2, \ldots, x_k] \).

For example

\[
f[x_2, x_3] = \frac{f[x_3] - f[x_2]}{x_3 - x_2}
\]

\[
f[x_3, x_4, x_5] = \frac{f[x_4, x_5] - f[x_3, x_4]}{x_5 - x_3}
\]

Hence, we can get

\[
a_1 = f[x_0, x_1] \\
a_2 = f[x_0, x_1, x_2]
\]

with a little algebra, and so on.

With this notation we can write down (22) as

\[
p(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0) \cdots (x - x_{k-1}).
\]

The iterative structure can be made obvious in the following table:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>1st div diff</th>
<th>2nd div diff</th>
<th>3rd div diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_0 )</td>
<td>( f[x_0] )</td>
<td>( f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} )</td>
<td>( f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} )</td>
<td>( f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} )</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( f[x_1] )</td>
<td>( f[x_2, x_1] = \frac{f[x_2] - f[x_1]}{x_2 - x_1} )</td>
<td>( f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} )</td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_4 )</td>
<td>( f[x_4] )</td>
<td>( f[x_4, x_3] = \frac{f[x_5] - f[x_4]}{x_5 - x_4} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_5 )</td>
<td>( f[x_5] )</td>
<td>( f[x_5, x_4] = \frac{f[x_5] - f[x_4]}{x_5 - x_4} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An Example: Take \( x = -3, -2, \ldots, 3, 4 \). Let
\[
f(x) = x^6 - 2x^5 - 4x^4 + x^3 + x - 20.
\]
Let’s first generate a table. You need 8 points and thus 8 divided difference levels.

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>841</td>
</tr>
<tr>
<td>-2</td>
<td>34</td>
</tr>
<tr>
<td>-1</td>
<td>-23</td>
</tr>
<tr>
<td>0</td>
<td>-20</td>
</tr>
<tr>
<td>1</td>
<td>-23</td>
</tr>
<tr>
<td>2</td>
<td>-74</td>
</tr>
<tr>
<td>3</td>
<td>-71</td>
</tr>
<tr>
<td>4</td>
<td>1072</td>
</tr>
</tbody>
</table>

This is how the table will look:

| \( x \) | \( f(x) \) | \( f[x] \) | \( f[x,x] \) | \( f[x,x,x] \) | \( f[x,x,x,x] \) | \( f[x,x,x,x,x] \) | \( f[x,x,x,x,x,x] \) | \( f[x,x,x,x,x,x,x] \) |
|---|---|---|---|---|---|---|---|
| -3 | 841 | | | | | | |
| -2 | 34 | 375 | | | | | |
| -1 | -23 | 30 | -115 | | | | |
| 0  | -20 | -3 | -11 | -5 | | | |
| 1  | -23 | -24 | 6 | 1 | | | |
| 2  | -74 | 27 | 17 | 7 | | | |
| 3  | -71 | 570 | 181 | | | | |
| 4  | 1072 | | | | | | |

DD stands for divided difference.
Algorithm: Generates the coefficients in the Newton form of the polynomial.

In what follows we use the notation

\[ F_{i,1} = f[x_i, x_{i+1}] \]

Further, we define

\[ F_{i,j} \equiv \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}. \]

Note that \( F_{i,i} = f[x_0, x_1 \ldots x_i] \). The algorithm generates \( F_{0,0}, F_{1,1}, \ldots, F_{n,n} \), so that

\[ p(x) = \sum_{i=0}^{n} F_{i,i} \prod_{j=0}^{i-1} (x - x_j) \]

where \( f(x_0) \equiv F_{0,0}; f(x_1) = F_{1,0}; f(x_2) = F_{2,0}; \ldots; f(x_n) = F_{n,0}. \)

Input: \( x_0, x_1, \ldots, x_n \) distinct.
\( f(x_0), f(x_1), \ldots, f(x_n) \).

Output: \( F_{0,0}, F_{1,1}, \ldots, F_{n,n} \)

for \( i=1:n \)
    for \( j = 1:n \)
        \( F_{i,j} = \frac{[F_{i,j-1} - F_{i-1,j-1}]}{x_i - x_{i-j}} \)
    end
end
Output \( (F_{0,0}, \ldots, F_{n,n}) \)

End

7.4.1 Interpolation at Equally-Spaced Points using Newton Divided Differences

Things simplify considerably when we use Newton’s divided differences in the interpolation of a function: Assume that

\[ a = x_0 < x_1 < \ldots < x_n = b \]
\[ x_i = x_0 + ih \quad h = \frac{b - a}{n} \quad 0 \leq i \leq n. \]
Recall that
\[
\begin{align*}
f[x_i, x_{i+1}] &= \frac{f(x_{i+1}) - f(x_i)}{h} \\
f[x_i, x_{i+1}, x_{i+2}] &= \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2h^2}
\end{align*}
\]

We define the forward difference operator \( \Delta \). When applied to \( f(x) \),
\[
\Delta f = f(x + h) - f(x).
\]

Hence,
\[
\Delta^{k+1} f(x) = \Delta(\Delta^k f) \quad k = 0, 1, 2 \ldots
\]
where zeroth power of an operator is the identity operator.

Example
\[
\Delta^2 f = \Delta(\Delta f) = \Delta(f(x + h) - f(x)) = (f(x + 2h) - f(x + h)) - (f(x + h) - f(x)) = f(x + 2h) - 2f(x + h) + f(x)
\]
thus
\[
\Delta^k f = \sum_{i=0}^{k} \binom{k}{i} (-1)^i f(x + (k - i)h)
\]
where
\[
\binom{k}{i} = \frac{k!}{i!(k - i)!} \quad \binom{k}{0} = \binom{k}{k} = 1.
\]

Let’s get back to interpolation. Using the Newton formula,
\[
p(x) = f(x_0) + f[x_0, x_1](x-x_0) + \cdots + f[x_0, x_1, \cdots, x_n](x-x_0)(x-x_1) \cdots (x-x_{n-1}).
\]
We express 

\[ x = x_0 + \rho h, \]

where \( 0 \leq \rho \). Then

\[
[x_0, x_1 \cdots x_j](x - x_0)(x - x_1) \cdots (x - x_{j-1}) = \frac{\Delta^j f(x_0)}{j!} \rho (\rho - 1)(\rho - 2) \cdots (\rho - (j - 1)).
\]

polynomial of degree \( j \) in the variable \( \rho \).

Note that

\[
\binom{r}{j} = \frac{r(r-1)(r-2) \cdots (r-(j-1))}{j!}.
\]

So, for \( j \geq 1 \),

\[
\binom{r}{0} = 1, \quad \binom{r}{1} = r, \quad \binom{r}{2} = \frac{r(r-1)}{2}, \quad \binom{r}{3} = \frac{r(r-1)(r-2)}{6},
\]

etc.

So we can write

\[
p(x) = p(x_0 + \rho h) = f(x_0) + \Delta f(x_0) \binom{\rho}{1} + \Delta^2 f(x_0) \binom{\rho}{2} \cdots \Delta^n f(x_0) \binom{\rho}{n}.
\]

Hence,

\[
p(x_0 + \rho h) = \sum_{i=0}^{n} \Delta^i f(x_0) \binom{\rho}{i}.
\]

\[ \square \]

Newton’s forward Formula: appears tabulated in old books on functions and series and books dedicated on divided differences:
7.5 Remarks on Polynomial Interpolation

The problem with polynomial interpolation? it seems overkill to use a 99th degree polynomial to fit 100 data points (how about a 999th degree poly for one thousand points?) The main problem with using high degree polynomials is this: the linear algebraic equations underlying the equations \((x_0, p_0) = (x_0, f_0), (x_1, p_1) = (x_1, f_1), \ldots (x_n, p_n) = (x_n, f_n)\) can be written as

\[ Aa = f, \]

where \(A\) is a Vandermonde matrix (see 17). This linear algebraic problem will become more and more ill-conditioned the large \(n\) is. In fact, it will not take too large a problem for it to become chronically ill-conditioned.

This is particularly troublesome if the data comes from an experiment, which means that the data is very noisy: this will force the coefficients contained in the vector \(a\) associated with high powers in \(x\) to be non-negligible. In the homework you get to explore how high degree polynomial interpolation can lead to unusual results.

However, there is a type of polynomial interpolation that is particularly well suited for data that is periodic, have knots at equally-spaced intervals, and for which the data does not include indefiniteness of any kind. For these
trigonometric polynomials are well suited. We will cover this, but before doing so we will consider a different strategy to polynomial interpolation:

A different approach, favored now in modern computing, is to use piecewise low order polynomials: \textit{i.e.}, divide the set of points into groups, fit these few points with a low order polynomial, and then glue these sections together. The error is then controlled by changing the order of the piecewise polynomials and/or by controlling the interval distances between points (when possible).

Two of these low order methods we’ll consider are “piece-wise linear polynomial interpolation,” which uses linear functions to connect the points, and “cubic splines.”

Splines are thin elastic rod like rulers drafters use. In the old days the splines were made of pliable lead, coated with cloth. Imagine trying to draft the shape of the hull of a ship. First, you want the shape to be symmetric about the centerline; second, you would like the hull to be made of the simplest smooth curve, that interpolates points on the curve which you nailed down. These points are called "knots." Once you’ve worked out the curve on one side of the hull, you just flipped the spline and could then draw the other side symmetrically. (Why these hulls took on these shapes are subjects matter for a fluid mechanics course).

Now suppose you wanted to use the lowest possible polynomial interpolators for a bunch of points. If you used piecewise constants, the figure would look like a bunch of blocks, side by side. If you use piecewise linear, the figure would look like a bunch of straight lines connecting at the edges. If the points are sufficiently dense, the figure would not look too badly. If you use piecewise parabolic, the figure would look odd…the problem is that quadratic functions do not support inflection points: so things would curve one way or the other, between points. By the eye-ball metric, the lowest order piecewise interpolator that looks natural is cubic splines.

Hence, we will study the piecewise linear (it is important and it is very fast to compute), and the cubic spline case, for the reason given above.

The cubic splines will not deliver an error estimate that is as low generally as the polynomial case, but it is computationally fast and stable and these two aspects will be more important when computing with finite-precision,
finite-time computing machines.

7.6 Spline Interpolation

Previously we discussed how to compute an approximation of a function over some finite interval \([a, b]\) using a single polynomial. Ignoring the practical issue of the inherent ill-conditioning of using high degree polynomials, attaining very high order accuracy was achieved by increasing the degree of the interpolant polynomial. An alternative strategy to obtain more accuracy, of course, is to use more points. If we are able to use more points, but very low order polynomials for each segment and then patch these up, we can circumvent the ill-conditioning problem and yet obtain good accuracy. Moreover, if we have good linear algebraic solvers (fast and efficient), we also get a high degree of efficiency in this procedure.

We will cover piece-wise linear and cubic splines. First, the piece-wise linear case. Piecewise linear are very simple and useful because of their simplicity and the resulting efficiency in obtaining an interpolation. They are especially useful in problems which require their integration, when the data set is huge, when a low accuracy interpolation is adequate. Piece-wise constant splines are used as well (for example in low order Riemann solvers in hyperbolic partial differential equations). Parabolic splines are not all that useful: they lack inflection points and thus their utility is very restricted. Cubic is then the next order up, and is the most common of the splines, especially in graphics.

Preliminaries:

Impose a subdivision of the interval \(I = [a, b]\) as follows:

\[
D \equiv a = x_0 < x_1 < x_2 < x_3, \ldots, x_{n-1} < x_n = b.
\]

Use a low-degree polynomial on each subinterval \([x_i, x_{i+1}]\), \(i = 0, 1, 2, \ldots, n-1\), patching these together using smoothness constraints.

\[
\text{Let } \Delta x_i \equiv x_{i+1} - x_i,
\]

i.e. the grid points may or may not be equally spaced.

For convenience, define \(|\Delta| \equiv \max_{1 \leq i \leq n-1} \Delta x_i\), the largest subinterval.
Again, we can control accuracy by choosing $|\Delta|$ to be as small as is needed. This, under the assumption that the local error is mainly concentrated where the grid points are most distant. This, of course, is not the entire story, as it also depends on the spectrum of the data itself.

Let $\mathcal{P}_m$ be the family of polynomials of degree at most $m$. We define a class of functions

$$S^k_m(\Delta) = \left\{ s : s \in C^k[a,b], s_{[x_i,x_{i+1}]} \in \mathcal{P}_m, i = 1, 2, \ldots, n - 1 \right\}$$

where $m \geq 0, k \geq 0$. $S^k_m(D)$ the “spline functions of degree $m$ and smoothness class $k$” relative to the subdivision $D$.

The continuity assumption of (24) is that the $k$th derivative of $s$ is continuous everywhere on $[a,b]$, and in particular, at the grid points $x_i$, ($i = 1, 2, 3, \ldots, n - 1$) of $D$.

N.B. If $k = m$ then $s \in S^m_m$ consist of 1 polynomial of degree $m$ and the whole interval $[a,b]$. We don’t want this, so $k < m$.

### 7.6.1 Piecewise Linear Case

Here we want $s \in S^0_1(D)$ such that for $f$ on $[a,b]$ $s(x_i) = f_i$; where $f_i = f(x_i), \ i = 1, 2, \ldots, n$.

We will assume that the interpolation points coincide with the grid locations $x_i$, although this is not a requirement.

Here, take $s_1(f; x) = f_i + (x - x_i) \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$ for $x_i \leq x \leq x_{i+1}, \ i = 1, 2, 3, \ldots, n - 1$. This is shown schematically in Figure 40.

What is the error? First recall from polynomial interpolation that for $x \in [x_0, x_1]$ the error for polynomial interpolators of degree 1 is

$$f(x) - p_1(x) = \frac{1}{2!}f''(\xi(x)) \prod_{i=0}^{1}(x - x_i),$$

for $x \in (x_0, x_1)$. This can be expressed equivalently as

$$f(x) - p_1(x) = f[x_0, x_1, x] \prod_{i=0}^{1}(x - x_i),$$
for $x \in (x_0, x_1)$.

The expression for the linear interpolator can also be written in terms of Newton divided differences as

$$s_1(f; x) = f_i + (x - x_i)f[x_i, x_{i+1}].$$

If $f \in C^2[a, b]$ the error is then

$$f(x) - s_1(f; x) = (x - x_i)(x - x_{i+1})f[x_i, x_{i+1}, x], \quad x \in [x_i, x_{i+1}]$$

$$|f(x) - s_1(f; x)| \leq \frac{(\Delta x_i)^2}{8} \max_{x \in [x_i, x_{i+1}]} |f''|, \quad \text{with } x \in [x_i, x_{i+1}]$$

thus

$$||f(x) - s_1(f; x)||_\infty \leq \frac{1}{8} |\Delta|^2 ||f''||_\infty$$

Which shows that the error can be made arbitrarily small by choosing the $\Delta x_i$'s small: the cost is the volume of data.

A Basis for $\mathcal{S}_i^0(\Delta)$: How many degrees of freedom do we have? We have 2 per segment and there are $n - 1$ segments. Hence

$$\dim \mathcal{S}_i^{-1}(D) = 2n - 2$$

140
Each continuity requirement adds 1 equation reducing the degrees of freedom by 1 per segment. Continuity at all interior points \( x_i, i = 2, 3, \ldots n - 1 \) eliminates the free parameter. Therefore,

\[
\dim \mathcal{S}_1^\circ (D) = 2n - 2 - (n - 2) = n.
\]

So the basis of the space \( \mathcal{S}_1^\circ (D) \) consists of \( n \) basis functions.

The Basis Functions: A convenient but not unique set of basis functions are defined as

\[
B_i(x) = \begin{cases} 
\frac{x-x_{i-1}}{x_i-x_{i-1}} & \text{if } x_{i-1} \leq x \leq x_i \\
\frac{x_{i+1}-x}{x_{i+1}-x_i} & \text{if } x_i \leq x \leq x_{i+1} \\
0 & \text{otherwise}
\end{cases}
\]

We’ll use slightly different looking basis functions at the two ends, at \( i = 1 \) and \( i = n \): At \( i = 1 \), \( B_1(x) = \frac{x-x_1}{x_2-x_1} \), and at \( i = n \), \( B_n(x) = \frac{x-x_{n-1}}{x_n-x_{n-1}} \).

These \( B_i \)’s are called “hat” functions, and an illustration of them appear in Figure 41.

We expect that these \( B_i(x) \) form a basis of \( \mathcal{S}_1^\circ (\Delta) \).

To prove this we must show

(i) \( \{B_i\}_{i=1}^n \) are linearly independent

(ii) They span the space \( \mathcal{S}_1^\circ (\Delta) \)

Proof: Use \( B_i(x_j) = \delta_{ij} \) to show this is true.

Application: Least-Squares Approximation: The \( L^2 \) approximation of data given at \( x_i, i = 1, 2, 3, \ldots, n \), corresponding to \( a \leq x_i \leq b \) is of course trivial. Suppose we want to find the interpolated value at any arbitrary \( x \in [a, b] \):
Given $f \in C[a, b]$, find $\hat{s}_1(f; \cdot) \in S_1^a(D)$ such that

$$\int_a^b |f(x) - s_1(f; x)|^2 dx \leq \int_a^b [f(x) - s(x)]^2 dx \quad \forall s \in S_1^a(\Delta) \quad (25)$$

Writing $\hat{s}(f; x) = \sum_{i=1}^{n} \hat{c}_iB_i(x)$ from (25)

$$\hat{s}_1(f; x) - f(x) = E$$
$$\sum_{i=1}^{n} \hat{c}_iB_i(x) - f(x) = E$$

where $E$ is the error. Next, multiply both sides by $B_j(x)$ and integrate:

$$\sum_{i=1}^{n} \left[ \int_a^b B_i(x)B_j(x)dx \right] \hat{c}_i \approx \int_a^b B_i(x)f(x)dx \quad i = 1, 2, 3, \ldots, n. \quad (26)$$

$B_i$ is nonzero on $(x_i, x_{i+1}) \Rightarrow \int_a^b B_i(x)B_j(x)dx = 0 \quad \text{if} \ |i - j| > 1.$
The resulting system of equations generates a matrix which is tridiagonal. An easy calculation (integrate (26)) shows that

\[ \frac{1}{6} \Delta x_{i-1} \tilde{c}_{i-1} + \frac{1}{3} (\Delta x_{i-1} + \Delta x_i) \tilde{c}_i + \frac{1}{6} \Delta x_i \tilde{c}_{i+1} = b_i; \]  

(27)

\[ i = 1, 2, 3, \ldots, n \]

Note that \( \Delta x_0 = 0 \quad \Delta x_n = 0 \), by convention.

The matrix problem in (27) is symmetric positive definite and easy to solve using the algorithm \((12.1.7)\).

\[ \square \]

### 7.6.2 Cubic Splines

This is the most popular piece-wise interpolation technique. Recall that the methodology in splining is to use low order polynomials to interpolate from grid point to grid point. This is ideally suited when one has control of the grid locations and the values of the data being interpolated (i.e. perhaps if you have a way to produce them at any location). In this case we can control the accuracy by making the grid spacing sufficiently small.

Why cubic splines? Because cubic splines are the lowest order polynomial endowed with inflection points. Think about interpolating a set of data points using parabolic (quadratic) functions: without inflection points the interpolation can look rather strange. Why not slightly higher order? because it is more expensive and the eye cannot really discern an appreciable difference between the cubic and higher order interpolating splines, provided the data set is sufficiently well distributed over the interval.

Here we could have used the method presented in connection with the piece-wise linear splines to construct the spline interpolation of the data using cubics. However, we purposely show an alternative technique, which is less elegant but nevertheless revealing. We use the most mundane and common technique in numerical methods: generate \( n \) linearly independent equations for \( n \) unknowns. Higher-order splines can be constructed similarly, however, the process becomes tedious.

Let \( S_p^k_n(x) \) be an \( n + 1 \)-member family of spline functions, where \( k \) is the
polynomial degree of the splines. Consider the cubic splines $S_n^3(x)$, which satisfy these properties:

$$\begin{align*}
S_n^3(x) & \in C^2[a, b] \text{ i.e. } S_n^3(x), \partial_x S_n^3(x), \partial_x^2 S_n^3(x) \text{ continuous on } [a, b] \\
S_n^3(x_j) & = f(x_j) \quad 0 \leq j \leq n \quad \text{i.e. } S_n^3(x) \text{ interpolates } f(x) \text{ on } [a, b] \\
S_n^3(x) & \text{ is a cubic polynomial on each interval } [x_j, x_{j+1}] \quad 0 \leq j \leq n - 1.
\end{align*}$$

(28)

So on the interval $[x_0, x_3]$ $S_n^3(x)$ is represented by the cubic polynomials $S_0^3$ on $[x_0, x_1]$, $S_1^3$ on $[x_1, x_2]$ and $S_2^3$ on $[x_2, x_3]$.

Since $S_j^3(x) \in P_3$ it has 4 coefficients. However, we would like it to satisfy (28) therefore we have 12 coefficients to pin down (count them).

Since each $S_j^3(x) \in P_3$, its derivatives are continuous for any open interval $x \in (x_j, x_{j+1})$, therefore we need to focus on obtaining continuity at each node $x_j$ location, where cubics will “patch together” with second order continuity.

Without loss of generality, $[a, b] \equiv [x_0, x_3]$. Here is how we can generate 12 equations for the 12 unknowns.

Six of the equations are generated by the conditions

$$\begin{align*}
S_0^3(x_0) & = f_0 \\
S_1^3(x_1) & = f_1 \\
S_2^3(x_2) & = f_2 \\
S_3^3(x_3) & = f_3.
\end{align*}$$

Since $S_n^3(x)$ and its first two derivatives are continuous at $x_1$ and $x_2$ we can get four more equations:

$$\begin{align*}
\partial_x S_0^3(x_1) & = \partial_x S_1^3(x_1) \\
\partial_x S_1^3(x_2) & = \partial_x S_2^3(x_2)
\end{align*}$$

All told we have 10 equations thus we expect an infinite number of solutions to the system of equations, i.e. a family of 2-parameter solutions. We can obtain 2 more equations by either assuming that

$$\begin{align*}
\partial_x^2 S_0^3(x_0) & = \partial_x^2 S_3^3(x_3) = 0 \\
& \text{ (“free” or “natural” boundary condition).} \\
\partial_x S_0^3(x_0) & = \partial_x f_0 \quad \text{and} \quad \partial_x S_3^3(x_3) = \partial_x f_3 \\
& \text{ (“clamped” boundary condition).}
\end{align*}$$
Generalizing, the complete list of properties is

\[(29) \quad S^3_j \text{ is a cubic polynomial on } [x_j, x_{j+1}] \quad j = 0 \cdots n - 1.\]

\[(30) \quad S^3_j(x_j) = f_j \quad j = 0 \cdots n \]

\[(31) \quad S^3_{j+1}(x_{j+1}) = S^3_j(x_{j+1}) \quad j = 0 \cdots n - 2 \]

\[(32) \quad \partial_x S^3_{j+1}(x_{j+1}) = \partial_x S^3_j(x_{j+1}) \quad j = 0 \cdots n - 2 \]

\[(33) \quad \partial^2_x S^3_{j+1}(x_{j+1}) = \partial^2_x S^3_j(x_{j+1}) \quad j = 0 \cdots n - 2, \]

plus one of the following is satisfied:

(i) \(\partial^2_x S^3(x_0) = \partial^2_x S^3(x_n) = 0\), free, OR

(ii) \(\partial_x S^3(x_0) = \partial_x f_0\) and \(\partial_x S^3(x_n) = \partial_x f_n\), clamped.

Note that for the clamped case we need derivative information: either from the properties of \(f(x)\) at the ends or by making an assumption.

Exercise: apply above conditions on

\[(34) \quad S^3_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3, \]

to construct the splines. Use \(h_j = x_{j+1} - x_j\), with \(j = 0, ..., n - 1\), and solve for \(c \equiv [c_0, c_1, ..., c_n]\). Show that for the natural end conditions, i.e. \(\partial^2_x S^3(a) = \partial^2_x S^3(b) = 0\), the full family of splines is given by the solution to the system

\[Ac = g\]

\(A\) is \((n + 1) \times (n + 1)\) matrix given by

\[
A = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 2(h_0 + h_1) & h_1 & 0 & 0 & 0 \\
0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & 0 \\
0 & 0 & \ddots & \ddots & \ddots & 0 \\
0 & 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
c = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \quad g = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}

Similarly, for the \textit{clamped case}, \( \partial_x S^3(a) = \partial_x f(a) \) and \( \partial_x S^3(b) = \partial_x f(b) \),

\[ Ac = g \]

is explicitly given by

\[ A = \begin{bmatrix} 2h_0 & h_0 & 0 & 0 & 0 & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix} \]

\[ c = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} \quad g = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix} \]

\[ \square \]

\textbf{Algorithm} The Natural Cubic Spline

\textbf{Input} \( n; x_0, \ldots, x_n, \)
\textbf{a}_0=f(x_0), \ldots, \textbf{a}_n=f(x_n)
\textbf{Output:} \( \textbf{a}_j, \textbf{b}_j, \textbf{c}_j \text{ d}_j \) for \( j=0:n-1 \)

\textbf{for} \( i=1:n-1 \)
\begin{verbatim}
    \textbf{h}_i=x_(i+1)-x_1
\end{verbatim}
\textbf{end}
for i=1:n-1
    alpha_i=(3/h_i)*(a_{i+1}-a_i)-(3/h_{i-1})*(a_i-a_{i-1})
end
l_0=1
mu_0=0
z_0=0
for i=1:n-1
    l_i=2*(x_{j+1}-x_{i-1})-h_{i-1}*mu_{i-1}
    mu_i= h_i/l_i
    z_i=(alpha_i-h_{i-1}*z_{i-1})/l_i
end
l_n=1
x_n=0
c_n=0
for j=n-1:-1:0
    c_j=z_j-mu_j*c_{j+1}
    b_j=(a_{j+1}-a_j)/h_j-h_j(c_{j+1}+2*c_{j})/3
    d_j=(c_{j+1}-c_j)/(3*h_j)
end
output (a_j, b_j, c_j, d_j for j=0:n-1)
end

Algorithm Clamped Cubic Spline Case

input n; x_0,...,x_n,
a_0=f(x_0), ..., a_n=f(x_n);
FPO=f'(x_0)
FPN=f'(x_n)

output: a_j, b_j, c_j, d_j for j=0:n-1
for i=0:n-1
    h_i=x_{i+1}-x_i
end
alpha_0=3*(a_1-a_0)/h_0-3*FPO
alpha_n = 3*FPN -3*(a_n-a_{n-1})/h_{n-1}
for i=1:n-1
    alpha_i=(3/h_i)*(a_{i+1}-a_i)-(3/h_{i-1})*(a_i-a_{i-1})
end
l_0=2*h_0
mu_0=0.5
z_0=alpha_0/l_0
for i=1:n-1
    l_i=2*(x_{i+1}-x_{i-1})-h_{i-1}*mu_{i-1}
    mu_i= h_i/l_i
    z_i=(alpha_i-h_{i-1}z_{i-1})/l_i
end
l_n=h_{n-1}*(2-mu_{n-1})
z_n=(alpha_n-h_{n-1}z_{n-1})/l_n
c_n=z_n
for j=n-1:-1:0
    c_j=z_j-mu_j*c_{j+1}
    b_j=(a_{j+1}-a_j)/h_j-h_j(c_{j+1}+2*c_j)/3
    d_j= (c_{j+1}-c_j)/(3*h_j)
end
output (a_j, b_j, c_j, d_j for j=0:n-1)
end

Figures 42 and 43 illustrate how cubic a spline interpolation compares with a
Lagrange polynomial interpolation. It also shows the effect of using different
conditions on the ends of the interval in the spline interpolation. In both cases
we are using the same knots. The derivative information for the clamped
spline case was easy to derive from the function itself. However, we emphasize
that this information is not always readily available. The result of the free
conditions at the end points for the cubic spline clearly show an effect on the
error near the end points.
Figure 42: Approximation of \( f(x) = x \sin(2\pi x + 1) \) using a sixth order Lagrange polynomial.

### 7.7 Trigonometric Interpolation

#### 7.7.1 Fourier Series Review

Definition: A function \( f = f(x) \) over \( \mathbb{R} \) is said to be periodic with period \( T > 0 \) if \( f(x + T) = f(x) \forall x \). The smallest period is the fundamental period.

If \( f(x) \) is periodic, and \( \int_{-T/2}^{T/2} |f(x)|^2 dx < \infty \), then we can write

\[
f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[ a_k \cos(k\omega x) + b_k \sin(k\omega x) \right]
\]

where \( \omega = \frac{2\pi}{T} \). The coefficients \( a_k \)'s and \( b_k \)'s can be found by

\[
a_k = \frac{1}{T} \int_{-T/2}^{T/2} f(x) \cos(k\omega x) dx \quad b_k = \frac{1}{T} \int_{-T/2}^{T/2} f(x) \sin(k\omega x) dx.
\]

We can also write:

\[
f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ik\omega x}
\]

149
Figure 43: Approximation of $f(x) = x \sin(2\pi x + 1)$ using cubic splines. The first figure uses the natural cubic spline end conditions. The second figure uses the clamped cubic spline conditions.
where the complex coefficients are defined by
\[ c_k = \frac{a_k - ib_k}{2} \quad \text{and} \quad c_{-k} = \frac{a_k + ib_k}{2}. \]

Without loss of generality, in what follows we will scale \( x \) so that the period is \( T = 2\pi \) in the new variable: let
\[ \tilde{f}(\tilde{x}) = f(x), \quad \tilde{x} = \frac{2\pi}{T}x. \]
Furthermore, we drop the tildes and the hats from the scaled independent and dependent variables, respectively.

Consider \( 2\pi \)-periodic functions which are
\[ \int_{-\pi}^{\pi} |u(x)|^2 \, dx < \infty. \]
we say that \( u \in L^2(-\pi, \pi) \), i.e., belonging to the space of \( L^2 \)-periodic functions, with period \( 2\pi \). In this space the norm is given in terms of the inner product as
\[ (u, v) = \int_{-\pi}^{\pi} \overline{u(x)} v(x) \, dx \]
here \( \overline{u} \) is the complex conjugate of \( u \).

**Recall:** if \( X \) is a linear vector space over \( \mathbb{C} \), then a map
\[ u, v \in X \rightarrow (u, v) \in \mathbb{C} \]
such that

(i) \( (u, v) = (\overline{u}, v) \) (symmetry),

(ii) \( (u, \lambda v) = \lambda(u, v) \) (scaling) \( \lambda \in \mathbb{C} \), a number

(iii) \( (u, v + w) = (u, v) + (u, w) \) (linear in second term)

(iv) \( (u, v) \geq 0 \) and \( (u, u) = 0 \) implies \( u = 0 \).
if $(i) - (iv)$ holds then $(\cdot, \cdot)$ is called an inner product on $X$.

Fact: if $(\cdot, \cdot)$ is an inner product of $X$ then

$$||u|| = (u, u)^{1/2}$$

and $|| \cdot ||$ is a norm on $X$.  \hfill \square

Two facts:

$u, v \in L^2$ are orthogonal if

$$(u, v) = 0$$

A useful inequality: “Cauchy Schwarz inequality” reads

$$|(u, v)| \leq ||u|| ||v||$$

\hfill \square

Let $\phi_k(x) \equiv e^{ikx}$, $k \in \mathbb{Z}$ then $(\phi_k, \phi_j) = \begin{cases} 2\pi & \text{if } k = j \\ 0 & \text{if } k \neq 0 \end{cases}$

Claim:

$$u(x) = \sum_{k=-\infty}^{\infty} \hat{u}_k \phi_k(x),$$

is a Fourier series representation of $u$, where

$$\hat{u}_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-ikx} u(x) dx = \frac{1}{2\pi} (\phi_k, u)$$

are the Fourier coefficients.

Important Fact: if

$$\sum_{k=-\infty}^{\infty} |\hat{u}_k| < \infty$$

$u(x)$ would be continuous by Lebesgue dominated convergence.
This is not true in general for $u \in L^2$. What is true is that if

$$\sum_{k=-\infty}^{\infty} |\hat{u}_k|^2 < \infty$$

then

$$2\pi \sum_{k=-\infty}^{\infty} |\hat{u}_k|^2 = ||u||^2_{L^2}.$$  

This is known as Parseval’s Identity.

In fact a Fourier series does not in general converge pointwise, but rather, in the $L^2$ sense: define

$$s^{n,m}(x) = \sum_{k=-n}^{n} \hat{u}_k \phi_k(x)$$

as the partial sum. Then

$$\lim_{n,m \to \infty} s^{n,m}(x) = u(x) \text{ in } L^2,$$

or

$$\lim_{n,m \to \infty} ||s^{n,n}(x) - u||_{L^2} = 0.$$  

(Look up convergence of the Fourier series in any advanced calculus book).

A classic example that illustrates how the Fourier series converges for non-smooth functions is the Fourier reconstruction of a step function. As you might recall the Fourier representation produces the Gibbs phenomenon, which is the appearance of concentrated oscillations at the edges of the step. As the number of Fourier terms is increased, the oscillations get more concentrated and stronger at the edges, but do not disappear. So the Fourier series does not converge uniformly. Try this out and convince yourself of this.

A very convenient feature of Fourier series is the following. The Fourier representation of the derivative of a function is computed quite simply:

$$\widehat{(\partial_x u)}_k = ik\hat{u}_k$$

$$\widehat{(\partial^n_x u)}_k = (ik)^n\hat{u}_k$$
So if \( u \in H^1 \) (i.e. \( u \in L^2 \) and \( \partial_x u \in L^2 \), in particular, no delta functions), by Parseval’s Identity:

\[
\int |\partial_x u|^2 \, dx = 2\pi \sum_{k=-\infty}^{\infty} k^2 |\hat{u}_k|^2 < \infty.
\]

If we had \( n \) derivatives in \( L^2 \) then

\[
\int |\partial^n_x u|^2 \, dx = 2\pi \sum_{k=-\infty}^{\infty} k^{2n} |\hat{u}_k|^2 < \infty.
\]

Suppose that further conditions are placed on the function and its derivatives. Could we get convergence in other norms?

\( \sum_{k=-\infty}^{\infty} |\hat{u}_k| < \infty \) would be really good convergence: let us further examine the case when \( u \in H^1 \). Hence

\[
\sum_{k \in \mathbb{Z}} \frac{(1 + k^2)^{1/2}}{(1 + k^2)^{1/2}} |\hat{u}_k| = \sum_{k \in \mathbb{Z}} \frac{1}{(1 + k^2)^{1/2}} (1 + k^2)^{1/2} |\hat{u}_k|.
\]

The term

\[
\sum_{k \in \mathbb{Z}} \frac{(1 + k^2)^{1/2}}{(1 + k^2)^{1/2}} |\hat{u}_k|
\]

converges and

\[
\sum_{k \in \mathbb{Z}} \frac{1}{(1 + k^2)^{1/2}} (1 + k^2)^{1/2} |\hat{u}_k|
\]

converges because \( u \in H^1 \). Hence, the Fourier series converges uniformly. So if \( u \in H^n \) then the Fourier series converges uniformly for the \( n - 1 \) derivative.

If \( u(x) \) is smooth, the Fourier series converges rapidly. In fact, we get what is known as spectral convergence. However, if \( u \) is not smooth the convergence can be very slow.

One place on the domain where this might be a problem in practical applications at \( x = -\pi \) and \( x = \pi \) when the function we are dealing with is not periodic.
Recall that the Fourier series reconstruction of a function defined over the period, here \([0, 2\pi]\), produces a periodic extension of the function, i.e. the function is defined over the whole real line and is \(2\pi\)-periodic. Reconstruction of non periodic functions yields slow (or just plain bad) convergence.

We can improve the convergence of a Fourier series reconstruction by constructing periodic extensions of a function that is not \(2\pi\) periodic.

In Figure 44 we illustrate a periodic extension that is periodic everywhere except at the end points. We want to avoid this to improve the convergence of the reconstruction.

We have to use our creativity to construct good periodic extensions and these depend on the shape of the function. Here are some examples.

A) Suppose \(u(-\pi) = u(\pi) = 0\) but not periodic: extend as in Figure 45. The periodic extension here is \(-u(x - 2\pi)\). The resulting function now has period \(2\pi\).

B) Suppose \(u(-\pi) = u(\pi) = \text{constant}\). In this case, use above trick after
subtracting the constant value out of the function. The constant is restored afterwards.

C) Suppose \( \partial_x u(-\pi) = \partial_x u(\pi) = 0 \). In this case use an even extension of \( u \) as in Figure 46. Again, resulting period is \( 4\pi \).

D) Mixed: \( u(-\pi) = 0, \partial_x(\pi) = 0 \). In this case the period would be \( 8\pi \).

\textbf{Exercise:} construct a periodized function, with period \( 4\pi \), with the above boundary conditions at \( \pm\pi \).

E) By zero padding if the function is compactly-supported with support smaller than \( 2\pi \), i.e. \( u \neq 0 \) in some continuous interval smaller than the period, and \( u = 0 \) (or a constant) otherwise. See Figure 47. The period remains \( 2\pi \). Here, zero-padding achieves an interpolation in Fourier space. Note that zero-padding in Fourier space achieves an interpolation in \( x \) space.

### 7.7.2 Interpolation using Fourier Polynomials

Consider periodic functions, that is, functions that satisfy

\[
f(t) = f(t + T), \quad \forall t \in (-\infty, \infty).
\]
We call $T$ the period. Without loss of generality we will consider $2\pi$-periodic functions
\[ f(t + 2\pi) = f(t). \]
(There is no difficulty considering larger or smaller $T$, just rescale to $2\pi$):

We can approximate $f(t)$ by a \textit{trigonometric polynomial}, as:

\begin{equation}
  p_n(t) = a_0 + \sum_{\ell=1}^{n} \left[ a_\ell \cos(\ell t) + b_\ell \sin(\ell t) \right]
\end{equation}

with $|a_n| + |b_n| \neq 0$.

For an $n^{th}$ degree trigonometric polynomial approximation of $f(x)$ we want $f(t_\ell) = p_n(t_\ell)$ for $\ell = 0, 1, 2, \ldots, 2n$ where
\[ 0 \leq t_0 < t_1 < t_2 \cdots < t_{2n} < 2\pi. \]

They will be equally spaced. We need $2n + 1$ points in $t$ since we need to fix $2n + 1$ coefficients in (35). Recall that $e^{i\theta} = \cos \theta + i \sin \theta$, hence (35) can be recast as
\[ p_n(t) = \sum_{\ell=-n}^{n} c_\ell e^{i\ell t} \]
where $c_0 = a_0$, $c_\ell = \frac{1}{2}(a_\ell - ib_\ell)$, $c_{-\ell} = \frac{1}{2}(a_\ell + ib_\ell)$, for $1 \leq \ell \leq n$. So, to determine the $\{c_\ell\}$’s we can find $\{a_\ell\}$ and $\{b_\ell\}$ (or vice versa).

Let $z = e^{it}$. Then
\[ p_n(z) = \sum_{\ell=-n}^{n} c_\ell z^\ell. \]
Thus, $z^n p_n(z)$ is a polynomial of degree $\leq 2n$.

Interpolation requires that
\begin{equation}
  p_n(z_\ell) = f(t_\ell) \quad \ell = 0, 1, \cdots, 2n.
\end{equation}
So we take $2n + 1$ distinct and equally-spaced points on the unit circle $|z| = 1$.

To see that (36) is uniquely solvable, note that (36) is equivalent to
\[ Q(z_\ell) = z_\ell^n f(t_\ell) \quad \ell = 0, 1, \cdots, 2n, \]

157
which is a polynomial interpolation problem with \(2n + 1\) distinct nodes \(z_0, z_1, \ldots, z_{2n}\). This can be used to establish uniqueness of the interpolating polynomial: Take \(S(z_\ell) = p_n(z_\ell) - Q(z_\ell)\) has degree \(\leq 2n\) and has \(2n + 1\) zeros, for \(\ell = 0, 1, \ldots, n\), hence \(S(z_\ell) := 0\).

Define a grid of points by

\[
(37) \quad t_\ell = \frac{2\pi}{2n + 1} \ell \quad \ell = 0, 1, \ldots, 2n.
\]

**Theorem:** Let \(\{t_\ell\}_{\ell=0}^{2n}\) be given by (37). Then the coefficients \(c_k\) of

\[
p_n(t_\ell) \equiv \sum_{k=-n}^{n} c_k e^{ikt_\ell}
\]

for \(\ell = 0, 1, \ldots, 2n\) are given by

\[
c_k = \frac{1}{2n + 1} \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell) \quad k = -n, \ldots, n.
\]

**Proof:** Want \(p_n(t_\ell) = f(t_\ell)\) for \(\ell = 0, 1, \ldots, 2n\). Hence

\[
\sum_{m=-n}^{n} c_m e^{imt_\ell} = f(t_\ell) \quad \text{for } \ell = 0, 1, \ldots, 2n.
\]

Multiply both sides by \(e^{-ikt_\ell}\) and sum over \(0 \leq \ell \leq 2n\),

\[
\sum_{\ell=0}^{2n} \sum_{m=-n}^{n} c_m e^{i(m-k)t_\ell} = \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell)
\]

and interchange the order of summations on the left hand side:

\[
(38) \quad \sum_{m=-n}^{n} c_m \left( \sum_{\ell=0}^{2n} e^{i(m-k)t_\ell} \right) = \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell).
\]

When \(m = k\)

\[
\sum_{\ell=0}^{2n} e^{i(m-k)t_\ell} = \sum_{\ell=0}^{2n} 1 = 2n + 1.
\]
When \( m \neq k \), first note that \( i(m - k)t_\ell = \frac{i\ell(m - k)2\pi}{2n + 1} \).

Let \( r = e^{i(m-k)2\pi/(2n+1)} \) and note \( r^{2n+1} = 1 \). Thus
\[
\sum_{\ell=0}^{2n} e^{i(m-k)t_\ell} = \sum_{\ell=0}^{2n} r^\ell = \frac{r^{2n+1} - 1}{r - 1} = 0.
\]

From (38),
\[
c_k = \frac{1}{2n + 1} \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell) \quad k = -n, \ldots, n.
\]

Further,
\[
\begin{align*}
a_\ell &= c_\ell + c_{-\ell} = \frac{1}{2n+1} \sum_{k=0}^{2n} (e^{-ikt_k} + e^{ikt_k}) f(t_k) = \frac{2}{2n + 1} \sum_{k=0}^{2n} f(t_k) \cos \ell t_k, \\

b_\ell &= i(c_\ell - c_{-\ell}) = \frac{2}{2n + 1} \sum_{k=0}^{2n} f(t_k) \sin \ell t_k
\end{align*}
\]

Example Find the trigonometric polynomial interpolation of degree 2 to 
\( f(t) = e^{\sin t + \cos t} \) on \([0, 2\pi]\):
\[
p_2(t) = \frac{a_0}{2} + c_1 \cos t + c_2 \cos 2t + b_1 \sin t + b_2 \sin 2t
\]
such that \( p_2(t_\ell) = e^{\sin t_\ell + \cos t_\ell} \quad t_\ell = \frac{2\pi}{5} \ell \) where \( \ell = 0, 1, 2, 3, 4 \).
\[
a_\ell = \frac{2}{2n + 1} \sum_{k=0}^{2n} f(t_k) \cos \ell t_k,
\]
\[
b_\ell = \frac{2}{2n + 1} \sum_{k=0}^{2n} f(t_k) \sin \ell t_k.
\]
\[
\Rightarrow a_0 = 3.12764 \quad a_1 = 1.24872 \quad a_2 = -0.09426 \quad b_1 = 1.27135 \quad b_2 = 0.49441.
\]

The function and the interpolation are shown in Figure 48.

\[\square\]
Figure 48: Exact (solid), versus interpolated (dashed) versions of the function, using 5 interpolation points.

8 NUMERICAL DIFFERENTIATION

Recall that the derivative of a function $f(x)$ at a point $x_0$ is given by

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}.$$  \hspace{1cm} (39)

Computing the quotient numerically does not seem to be such a big problem; however, an efficient and universal process by which we find an approximation to the limit is not clear.

Actually, as we will present later on, taking a quotient CAN be a problem on a computer: because of finite precision we can wind up with serious loss of precision errors. From the above formula it is evident that we may be subtracting nearby quantities, and further, the division of quantities which can be severely off-balance.

We will not cover differentiation and integration in a very deep and thorough way. The objectives are modest:
to appreciate the pitfalls that can come about in approximating derivatives (and later, in integrals),

• to learn simple Taylor series methods to developing approximations to derivatives and estimating their error.

• the most important lesson is to learn to use our knowledge of interpolation to develop approximations of derivatives (and differential operators in general).

We already saw one very efficient and robust way to obtain derivatives (and even integrals) of a special type of class of functions: for \(L_2\)-periodic functions we might use spectral methods instead, i.e. you use a Fourier interpolating polynomial approximation to the function, and take derivatives by multiplication (see 7.7).

### 8.1 Divided Differences/Taylor Series

Here we will present one way to obtain derivatives, by divided differences and Taylor series.

In what follows we’ll assume that

\[
x_0 \in (a, b), \quad f \in C^2[a, b]
\]

\[
x_1 = x_0 + h, \quad \text{with } h \neq 0 \quad \text{small so that } x_1 \in (a, b).
\]

Taking the first terms in the Taylor series, about \(x = x_0\),

\[
f(x_1) = f(x_0 + h) \approx f(x_0) + hf'(x_0).
\]

More precisely,

\[
f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{1}{2}h^2 f''(\xi), \quad \text{with } \xi \in (x_0, x_0 + h).
\]

Therefore

\[
f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{1}{2}hf''(\xi).
\]
Comparison of (39) and (40) shows that replacing the limiting procedure by a difference introduces an error, which we will call the truncation error. In this case the truncation error is
\[
\frac{1}{2}hf''(\xi) = O(h).
\]

Using (40), with \( h > 0 \), we obtain the “first order forward difference formula for the derivative”; when \( h < 0 \) we obtain the first-order backward difference formula. The order alludes to the order of the truncation error.

Let’s look at how this truncation error shows up:

**Example** Take \( f(x) = \ln x \), and first, compute the derivative at \( x_0 = 1.8 \), using several values of \( h \):

\[
\frac{f(1.8 + h) - f(1.8)}{h} \quad h > 0
\]

used to find \( f'(1.8) \) with error

\[
\frac{|hf''(\xi)|}{2} = \frac{|h|}{2\xi^2} \leq \frac{|h|}{2(1.8)^2} \text{ where } 1.8 < \xi < 1.8 + h
\]

\( \searrow \) from \( f'' \)

| \( h \) | \( f' \) | \( \frac{|h|}{2\xi^2} \) |
|-------|-------|----------------|
| 0.1   | 0.5406722 | 0.0154         |
| 0.01  | 0.5546180 | 0.00154        |
| 0.001 | 0.55554013 | 0.000154       |

A plot of the truncation error shows that the error drops by two whenever the size of \( h \) drops by two, say. The derivative of the same function but now at \( x_0 = 0.2 \) reveals an interesting aspect of the algorithm:

| \( h \) | \( f' \) | \( \frac{|h|}{2\xi^2} \) |
|-------|-------|----------------|
| 0.1   | 10.98612 | 1.25           |
| 0.01  | 304.4522 | 0.125          |
| 0.001 | 5303.305 | 0.0125         |
In both cases we see the error dropping inversely with the size of $h$. However, for $x_0 = 1.8$ we see the changes in the value of $f'$ are small and so is the error. For $x_0 = 0.2$, however, the opposite is true. It seems that it matters what $x_0$ one chooses in the evaluation of the derivative. This is indeed the case, but it also matters what function you are differentiating. This is an important aspect of numerical analysis called “sensitivity analysis” or “error analysis”. Later on we will discuss this in the context of “ill-conditioning”.

\[ \square \]

8.2 Derivatives via Lagrange Polynomial Interpolation

A way to systematically derive approximations to any order derivative, to whatever truncation error is desired, is to use Lagrange polynomial interpolation (you can do this very efficiently using Newton divided differences, actually, if the lattice over which you define an approximation to the function $f(x)$ which you plan to differentiate, is made of equally-spaced points).

It is a very general procedure, however, the issue of roundoff error is being ignored here completely by assuming that the quantities have infinite precision. We defer the issue of finite-precision to 8.2.1.

Suppose \( \{x_0, x_1, \ldots, x_n\} \) are \( n + 1 \) distinct numbers, \( I \) is an interval containing such numbers, and \( f \in C^{n+1}(I) \).

Using cardinal polynomials \( \ell \), we interpolate the function

\[
f(x) = \sum_{k=0}^{n} f(x_k) \ell_k(x) + \frac{(x-x_0) \cdots (x-x_n)}{(n+1)!} f^{(n+1)}(\xi) \]

for \( \xi(x) \in I \). The second term is the “error” term.

Differentiating:

\[
f'(x) = \sum_{k=0}^{n} f(x_k) \ell'_k(x) + \frac{d}{dx} \left[ \frac{(x-x_0) \cdots (x-x_n)}{(n+1)!} \right] f^{(n+1)}(\xi(x)) \\
+ \frac{1}{(n+1)!} (x_0 - x_0) \cdots (x - x_n) \frac{d}{dx} \left( f^{(n+1)}(\xi(x)) \right) \]

163
What is the truncation error?

If \( x = x_k \) then term involving \( D_x[f^{n+1}(\xi)] = 0 \). Therefore,

\[
f'(x_k) = \sum_{j=0}^{n} f(x_j)\ell'_j(x_k) + \frac{f^{(n+1)}(\xi(x_k))}{(n+1)!} \prod_{j=0}^{n} (x_k - x_j)
\]

which is equivalent to the \( n+1 \)-point formula, since

\[
\frac{\partial}{\partial x} \prod_{j=0}^{n} (x - x_j) = \sum_{j=0}^{n} \prod_{i=0 \atop i \neq j}^{n} (x - x_i), \quad \text{if} \ x = x_k \Rightarrow \prod_{j=0 \atop j \neq k}^{n} (x_k - x_j).
\]

One might think that the higher the truncation error, the more accurate the answer. In general, using more points leads to greater accuracy. However, higher orders mean more functional evaluations and greater potential of round-off error. Thus, we must balance truncation error versus round-off error and speed benefits. One could instead use a low order formula but use a finer \( h \) to obtain a desired accuracy, and in fact, in most applications, this is the way to go (provided you have the luxury of determining the discrete lattice.

Most common are forward and backward formulas. But other useful ones are the 3 and 5 point formulas:

\[
\ell_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \quad \ell'_0(x) = \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)},
\]

\[
\ell_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}, \quad \ell'_1(x) = \frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)},
\]

\[
\ell_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}, \quad \ell'_2(x) = \frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}.
\]

Therefore,

\[
f'(x_j) = f(x_0) \left[ \frac{2x_j - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)} \right] + f(x_1) \left[ \frac{2x_j - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)} \right] + f(x_2) \left[ \frac{2x_j - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)} \right] + \frac{1}{6} f^{(3)}(\xi_j) \frac{d^2}{dx^2} \prod_{i=0 \atop i \neq j}^{n} (x_j - x_i), \quad (41)
\]
for each $j = 0, 1, 2$ and $\xi_j = \xi(x_j)$.

Using (41) on an evenly spaced grid, we get the following formula: Let $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, with $h \neq 0$.

Take $x_j = x_0$, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$. Then,

$$f'(x_j) = \frac{1}{h} \left[ -\frac{3}{2} f(x_0) + 2 f(x_1) - \frac{1}{2} f(x_2) \right] + \frac{h^2}{3} f^{(3)}(\xi_0).$$

Now take $x_j = x_1$,

$$f'(x_1) = \frac{1}{h} \left[ -\frac{1}{2} f(x_0) + \frac{1}{2} f(x_2) \right] - \frac{h^2}{3} f^{(3)}(\xi_1).$$

Finally, when $x_j = x_2$,

$$f'(x_2) = \frac{1}{h} \left[ \frac{1}{2} f(x_0) - 2 f(x_1) + \frac{3}{2} f(x_2) \right] + \frac{h^2}{3} f^{(3)}(\xi_2).$$

Since $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, we can shift all of these so that we can compare them: we can translate to get

$$f'(x_0) = \frac{1}{2h} \left[ -3 f(x_0) + 4 f(x_0 + h) - f(x_0 + 2h) \right] + \frac{h^2}{3} f^{(3)}(\xi_0),$$

$$f'(x_0) = \frac{1}{2h} \left[ -f(x_0 - h) + f(x_0 + h) \right] - \frac{h^2}{6} f^{(3)}(\xi_1),$$

$$f'(x_0) = \frac{1}{2h} \left[ f(x_0 - 2h) - 4 f(x_0 - h) + 3 f(x_0) \right] + \frac{h^2}{3} f^{(3)}(\xi_2).$$

(These are really 2 formulas, since the last and first are equivalent by letting $h \to -h$). In summary:

\begin{align*}
(42) \quad f'(x_0) &= \frac{1}{2h} \left[ -3 f(x_0) + 4 f(x_0 + h) - f(x_0 + 2h) \right] + \frac{h^2}{3} f^{(3)}(\xi_0), \\
(43) \quad f'(x_0) &= \frac{1}{2h} \left[ f(x_0 + h) - f(x_0 - h) \right] - \frac{h^2}{6} f^{(3)}(\xi_1),
\end{align*}

where

- $\xi_0$ lies between $x_0$ and $x_0 + 2h$,
- $\xi_1$ lies between $(x_0 - h)$ and $(x_0 + h)$.
We note that the error in the centered formula, (43), is half of the forward formula, (42).

3-Point Second-order Forward and Centered Difference Formulas:

\[ f'(x_0) \approx \frac{1}{2h} [-3f(x_0)+4f(x_0+h)-f(x_0+2h)], \quad f'(x_0) \approx \frac{1}{2h} [f(x_0+h)-f(x_0-h)]. \]

An adequate approximation to the first derivative of \( f \) evaluated at \( x_0 \), provided \( h << 1 \) is sufficiently small. These are "second order" approximation formulas because the truncation errors are

\[ \frac{h^2}{3} f^{(3)}(\cdot) = O(h^2), \quad \text{and} \quad \frac{h^2}{6} f^{(3)}(\cdot) = O(h^2). \]

In a similar way one can derive

2-Point First-Order Forward (and Backward) Difference Formulas:

\[ f'(x_0) = \frac{1}{h} [f(x_0 + h) - f(x_0)] + \frac{h}{2} f''(\xi). \]  

A \( O(h) \)-order approximation is obtained by truncating the error.

5-Point Fourth-Order Forward and Centered Difference Formulas:

\[ f'(x_0) = \frac{1}{12h} [f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)] + \frac{h^4}{30} f^5(\xi) \]  

and

\[ f'(x_0) = \frac{1}{12h} [-25f(x_0) + 48f(x_0 + h) - 36f(x_0 + 2h) + 16f(x_0 + 3h) - 3f(x_0 + 4h)] + \frac{h^4}{5} f^5(\xi), \]

where \( x_0 \leq \xi \leq x_0 + 4h \). Use \( h < 0 \) above to obtain backward difference formula.
Why bother with the forward and backward formulas, when the centered ones have smaller truncation error? The forward formulas, (42) and (46), are useful at end of a finite interval. The backward formulas, obtained by letting \( h \to -h \) in the forward formulas, are useful at the beginning of the finite interval.

How to get higher-order derivatives? Can use same procedure as above.

**Taylor Series Strategy:**

Say you need to find the formula for the divided difference formula for \( f''(x_0) \):

We perform Taylor series expansions, assuming \(|h| \ll 1|:

\[
f(x_0 + h) = f(x_0) + f'(x_0)h + f''(x_0)\frac{h^2}{2} + f^{(3)}(x_0)\frac{h^3}{6} + f^{(4)}(\xi_h)\frac{h^4}{24},
\]

and

\[
f(x_0 - h) = f(x_0) - f'(x_0)h + f''(x_0)\frac{h^2}{2} - f^{(3)}(x_0)\frac{h^3}{6} + f^{(4)}(\xi_h)\frac{h^4}{24}.
\]

Adding these 2 expressions and solving for \( f'' \):

\[
f''(x_0) = \frac{1}{h^2}\left[f(x_0 - h) - 2f(x_0) + f(x_0 + h)\right] - \frac{h^2}{24}\left[f^{(4)}(\xi_h) + f^{(4)}(\xi_h)\right].
\]

If \( f^{(4)} \) is continuous on \([x_0 - h, x_0 + h]\) the Intermediate Value Theorem is invoked and thus,

\[
f''(x_0) = \frac{1}{h^2}[f(x_0 + h) - 2f(x_0) + f(x_0 - h)] - \frac{h^2}{12}f^{(4)}(\xi)
\]

for some \( x_0 - h < \xi < x_0 + h \).

**Second-order Center-difference Formula for the Second Derivative:**

If we approximate

\[
f''(x_0) \approx \frac{1}{h^2}[f(x_0 + h) - 2f(x_0) + f(x_0 - h)],
\]

167
we obtain the second order center-difference approximation to the second derivative of $f$ at $x_0$. It is "second order" because the truncation error

$$-\frac{h^2}{12} f^{(4)}(\xi) = O(h^2).$$

\begin{proof}
\end{proof}

8.2.1 The Effect of Round-off Error

Earlier we mentioned that roundoff error might be a problem in calculating derivatives using divided differences. For illustration, let us see how this shows up in the approximate calculation of a first derivative. So suppose we take into account truncation and round off errors, we will approximate $f'$.

Let $\tilde{f}$ be computed approximation to $f$ and we pick the center-difference formula for the calculation. The center difference formula yields

$$f'(x_0) = \frac{1}{2h} [f(x_0 + h) - f(x_0 - h)] - \frac{h^2}{6} f^{(3)}(\xi_1),$$

however, in the evaluation of the forward and backward terms we encounter round-off error $e(x_0 + h)$ and $e(x_0 - h)$, respectively. That is, $f(x_0 \pm h) = \tilde{f}(x_0 \pm h) + e(x_0 \pm h)$. Therefore

$$f'(x_0) - \frac{\tilde{f}(x_0 + h) - \tilde{f}(x_0 - h)}{2h} = \frac{e(x_0 + h) - e(x_0 - h)}{2h} - \frac{h^2}{6} f^{(3)}(\xi_1).$$

Let us assume that $|e(x_0 \pm h)| < \varepsilon$ where $\varepsilon > 0$ and $|f^{(3)}(\xi_1)| < M$ where $M > 0$ is some number. Then,

$$|f'(x_0) - \frac{\tilde{f}(x_0 + h) - \tilde{f}(x_0 - h)}{2h}| \leq \frac{\varepsilon}{h} + \frac{h^2}{6} M. \tag{47}$$

Hence, in order to reduce truncation error, we need to reduce $h$. However, $\frac{\varepsilon}{h}$ grows as $h$ gets smaller. Hence, making $h$ too small can make the total error be dominated by round-off (See (47)).
Figure 49: *Error in the central difference approximation to the derivative $f'(x)$ as a function of $h$. The dots correspond to the error solely due to truncation error. Stars correspond to the error from both truncation and roundoff.*
In Figure 49 we show the error as a function of $h$ for the central difference approximation of a smooth function. It is clear from the star curve, which corresponds to both the truncation and roundoff error contributions, that for small $h$ the error is dominated by the roundoff and for larger $h$ it is dominated by the truncation error. The optimal value of $h$, which minimizes the error, can be found by minimizing the expression for the error.

The table below shows the errors for different values of $h$. Note that for larger values of $h$ ($h > 10^{-6}$) the errors seem to be decreasing by a factor of approximately 100. This is due to the truncation error of the central difference method. As the method is of order $O(h^2)$ when we decrease $h$ by ten we decrease the error by 100. But as $h$ becomes smaller the roundoff error of the computer becomes more important and the error starts to increase with decreasing $\epsilon$. This increase is approximately by a factor of 10 with every decrease of $h$ by a factor of 10. This is because the $\frac{\epsilon}{h}$ term is dominating for small values of $h$. The value of $M$ given in this example is $f'''(x_0) = 4804.6$; the value of (inferred from the table) is $\epsilon$ is $2 \times f(x_0) \times$ machine precision = $2.4246 \times 10^{-14}$.

<table>
<thead>
<tr>
<th>$h$</th>
<th>Error</th>
<th>Estimated Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-1}$</td>
<td>8.124453</td>
<td>8.007729</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>8.008886 $\times 10^{-2}$</td>
<td>8.007729 $\times 10^{-2}$</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>8.007740 $\times 10^{-4}$</td>
<td>8.007729 $\times 10^{-4}$</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>8.007935 $\times 10^{-6}$</td>
<td>8.007971 $\times 10^{-6}$</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>8.172410 $\times 10^{-8}$</td>
<td>8.250193 $\times 10^{-8}$</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>6.761866 $\times 10^{-9}$</td>
<td>2.504722 $\times 10^{-8}$</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>2.348227 $\times 10^{-7}$</td>
<td>2.424725 $\times 10^{-7}$</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>1.194055 $\times 10^{-6}$</td>
<td>2.424645 $\times 10^{-6}$</td>
</tr>
<tr>
<td>$10^{-9}$</td>
<td>1.763533 $\times 10^{-5}$</td>
<td>2.424645 $\times 10^{-5}$</td>
</tr>
<tr>
<td>$10^{-10}$</td>
<td>7.233669 $\times 10^{-6}$</td>
<td>2.424645 $\times 10^{-4}$</td>
</tr>
<tr>
<td>$10^{-11}$</td>
<td>1.704020 $\times 10^{-4}$</td>
<td>2.424645 $\times 10^{-3}$</td>
</tr>
<tr>
<td>$10^{-12}$</td>
<td>1.757870 $\times 10^{-2}$</td>
<td>2.424645 $\times 10^{-2}$</td>
</tr>
<tr>
<td>$10^{-13}$</td>
<td>1.493988 $\times 10^{-1}$</td>
<td>2.424645 $\times 10^{-1}$</td>
</tr>
<tr>
<td>$10^{-14}$</td>
<td>2.230919</td>
<td>2.424645</td>
</tr>
<tr>
<td>$10^{-15}$</td>
<td>1.875648</td>
<td>24.24645</td>
</tr>
<tr>
<td>$10^{-16}$</td>
<td>218.3926</td>
<td>242.4645</td>
</tr>
</tbody>
</table>
9 NUMERICAL INTEGRATION

The general methodology in approximating Riemann integrals is to replace integrals by finite sums. This procedure is called *numerical Quadrature*.

The simplest of ideas is to write the approximation

\[ \int_a^b f(x)dx \approx \sum_{i=0}^n c_i f(x_i), \]

and then find \( n+1 \) equations for the \( n+1 \) unknown constants \( c_i, i = 0, 1, ..., n \).

This quadrature procedure makes sense since integration is a linear operation, and if we need to estimate the integral using only a finite number of evaluations of the integrand, we must use a linear combination of the function values.

The above formula suggests that in devising different quadrature strategies, that we can choose different criteria for the weights \( c_i \), the knots at which we require evaluations, i.e., the \( f(x_i) \), and the number of points we will choose, \( n \).

9.1 Quadrature Via Polynomial Approximation

One very natural way to do this is with Lagrange polynomials. We will assume that you are now familiar with Polynomial Approximation (see Section 7).

Assume \( \{x_i\}_{i=0}^n \) are \( n+1 \) distinct points. Then

\[ (48) \quad P_n(x) = \sum_{i=0}^n f(x_i)\ell_i(x). \]

Here \( \ell_i(x) \) are just the cardinal functions. Replacing,

\[ \int_a^b f(x)dx = \int_a^b \sum_{i=0}^n f(x_i)\ell_i(x)\,dx + \int_a^b \prod_{i=0}^n (x-x_i)\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\,dx \]

error in approximation of \( f(x) \)

---

\(^7\)Integration in more than one independent variable is sometimes called Cubature.
\[
\sum_{i=0}^{n} c_i f(x_i) + \frac{1}{(n+1)!} \int_{a}^{b} \prod_{i=0}^{n} (x - x_i) f^{(n+1)}(\xi(x)) dx
\]

where \( \xi(x) \in [a, b] \), for each \( x \). This then determines what the values of \( c_i \) are:

\[
c_i = \int_{a}^{b} \ell_i(x) \, dx, \text{ for each } i = 0, 1, \ldots, n.
\]

( Again, see Section 7). \( c_i \) are known as the \textit{quadrature weights}. We will mostly be using this formula in the case where the \( x_i \) are evenly distributed over the interval \([a, b]\). So,

\[
x_i = a + ih \quad \text{where } h = \frac{b - a}{n}
\]

In this case it is useful to make a change of variable when evaluating the integral of the \( \ell_i(x) \). Let \( x = a + hu \), then \( u = (x - a)/h \). Hence,

\[
c_i = \int_{a}^{b} \frac{\prod_{j\neq i} (x - x_j)}{\prod_{j\neq i} (x_i - x_j)} dx
\]

\[
= h \int_{0}^{h} \frac{\prod_{j\neq i} (u - j)}{\prod_{j\neq i} (i - j)} du
\]

\[
\equiv h \gamma_i
\]

Notice that the value of \( \gamma_i \) is a number which is independent of \( a \) and \( b \), but dependent on \( n \). The formulas for 1\textsuperscript{st} and 2\textsuperscript{nd} degree Lagrange polynomial interpolation at equally-spaced intervals generate the Trapezoidal Rule (see 9.1.1) and the Simpson’s Rule (see 9.2).

The way we will proceed is to figure out how to integrate over just enough quadrature points (2, for 1\textsuperscript{st} degree polynomials, and 3, for 2\textsuperscript{nd} degree polynomials). Once we figure out how to make the approximation over these small intervals we then use the linearity of integration,

\[
\int_{a}^{b} f(x) \, dx = \int_{X_0}^{X_1} f(x) \, dx + \int_{X_1}^{X_2} f(x) \, dx + \ldots + \int_{X_{m-1}}^{X_m} f(x) \, dx,
\]

172
where \( a = X_0 < X_1 < X_2 \ldots < X_m = b \), to write down composite quadrature formulas of integration, in terms of the \( m \) panels \( X_{j+1} - X_j \). These panels do not have to be of the same width. (See Section 9.5).

**9.1.1 The Trapezoidal Rule:**

Here we consider the case \( n = 1 \) in (48), i.e. linear approximation. The panel is of length \( h \). Without loss of generality, let \( h = b - a \). Then \( \gamma_0 = \int_0^1 \frac{u - 1}{0 - 1} du = \frac{1}{2} \), and \( \gamma_1 = \int_0^1 \frac{u - 0}{1 - 0} du = \frac{1}{2} \). This gives the trapezoidal rule approximation of the integral as

\[
\int_a^b f(x) \, dx \approx \frac{h}{2} (f(a) + f(b)).
\]

It is clear that we are approximating the (signed) area under \( f(x) \) as a trapezoidal box.

In order to estimate the error in this approximation, we need to estimate

\[
E(f) = \int_a^b \frac{(x - a)(x - b)}{2} f''(\xi(x)) \, dx.
\]

Using the change of variables as before \( (x = a + hu) \), it is convenient to rewrite this as

\[
E(f) = h^3 \int_0^1 \frac{u(u - 1)}{2} f''(\xi(a + uh)) \, du.
\]

We would like to be able to simplify this error term so that we do not have an integral of an unknown function. It is the \( \xi(x) \) term that causes the difficulty. We will use the Weighted Mean Value Theorem for this purpose.

**Theorem** (Weighted Mean Value Theorem)

Let \( f \in C[a, b] \), and \( g \) be integrable on \([a, b]\), with the additional constraint that \( g \) is either strictly positive or negative on \([a, b]\) then there exists a constant \( c \) in \((a, b)\) such that

\[
\int_a^b f(x)g(x) \, dx = f(c) \int_a^b g(x) \, dx
\]

173
Note: The reason for the name of the theorem is that the formula can be written as
\[ \bar{f} \equiv \frac{\int_{a}^{b} f(x) g(x) \, dx}{\int_{a}^{b} g(x) \, dx} = f(c) , \]
where \( \bar{f} \) is the weighted mean value (for cases where \( g \) is non-negative.)

Proof: The proof is instructive and similar to the Mean Value Theorem proof. We will restrict this proof to the case where \( g(x) \) is piecewise continuous and \( \int_{a}^{b} g(x) \, dx \neq 0 \) although the theorem is more general than this.

Suppose that no constant \( c \) exists. Let \( \bar{f} = \frac{\int_{a}^{b} f(t) g(t) \, dt}{\int_{a}^{b} g(t) \, dt} \). Since \( f(x) \) is a continuous function, this means that either \( f(x) < \bar{f} \) for all \( x \) in \( (a,b) \) or \( f(x) > \bar{f} \) for all \( x \) in \( (a,b) \). Without loss of generality, let us assume we are in the first case. Also, without loss of generality, assume \( g(x) \geq 0 \) on \( (a,b) \). Then, multiplying the two inequalities together we obtain \( f(x) g(x) \leq \bar{f} g(x) \).

Notice that if we had a strict inequality we would be done: just integrate from \( a \) to \( b \) and the resulting formula would be
\[ \int_{a}^{b} f(x) g(x) \, dx < \bar{f} \int_{a}^{b} g(x) \, dx \]
. Dividing through by the integral of \( g \) and recalling the definition of \( \bar{f} \) gives the contradictory conclusion that \( \bar{f} < \bar{f} \). Thus we would conclude that the original assumption that no \( c \) in \( (a,b) \) with \( f(c) = \bar{f} \) exists is not correct, and the theorem would be proved.

To get the strict inequality for the integral, we use the continuity of \( f(x) \), the assumption that \( \int_{a}^{b} g(x) \, dx > 0 \), and our assumption that \( g(x) \) is piecewise continuous. This allows us to conclude that \( g(d) > 0 \) at some point \( d \) in \( (a,b) \) where \( g(x) \) is continuous. (If not, then \( g(x) = 0 \) except at a finite number of points, and then the integral would have to be zero). At the point \( x = d \) we know that \( f(d) < \bar{f} \). Let \( \epsilon = (\bar{f} - f(d))/2 \). By continuity of \( f(x) g(x) - \bar{f} g(x) \) at \( d \), there is an interval \( (d - \delta, d + \delta) \subset (a,b) \) where \( f(x) g(x) \leq (f) g(x) - \epsilon \). Integrating from \( a \) to \( b \) and splitting the interval into pieces we get
\[
\int_{a}^{b} f(x) g(x) \, dx = \int_{a}^{d-\delta} f(x) g(x) \, dx + \int_{d-\delta}^{d+\delta} f(x) g(x) \, dx + \int_{d+\delta}^{b} f(x) g(x) \, dx
\]
\[
\leq f \left( \int_a^{d-\delta} g(x) \, dx + \int_{d+\delta}^{b} g(x) \, dx + \int_{d+\delta}^{b} g(x) \, dx \right) - 2\delta \epsilon \\
= \int_a^b g(x) \, dx - 2\delta \epsilon < \int_a^b g(x) \, dx.
\]

The case where \( \int_a^b g(x) \, dx = 0 \) is not hard to check, but since we will not need it, we will skip it.

Returning to our original problem of estimating the integral, we use the theorem as follows: Since \( u(u - 1) \) does not change sign on \([0, 1]\) and is integrable, we can say that

\[
h^3 \int_0^1 f''(\xi(a + hu)) \frac{u(u - 1)}{2} \, du = h^3 f''(\xi) \int_0^1 \frac{u(u - 1)}{2} \, du \\
= h^3 f''(\xi) \left[ \frac{u^3}{6} - \frac{u^2}{4} \right]_0^1 = -\frac{h^3}{12} f''(\xi)
\]

Putting it all together we have

\[
\int_a^b f(x) \, dx = \frac{h}{2} (f(a) + f(b)) - h^3 \frac{f''(\xi)}{12}
\]

where \( h = b - a \) and \( \xi \in (a, b) \).

Remark: The formula is exact if \( f'' = 0 \) (i.e. polynomial of degree 1 or less). This exactness on polynomials of degree less than or equal to 1, is (what we define to be) an order of accuracy of 1.

**Definition:** Quadrature rules are of the \( n^{th} \) order of accuracy, if they are exact in the approximation of an integral of an \( n^{th} \)-degree polynomial.

Schematically, the situation in the Trapezoidal rule is shown in Figure 50

**Exercise** An important fact about the trapezoidal rule is obtained by considering the approximation of the integral of \( f(x) \), a bounded and continuous function that happens to be periodic, i.e., \( f(x + T) = f(x) \) for all \( x \). We will make the estimate in class (or in the homework), but we get the remarkable result that the trapezoidal rule is "spectrally-accurate."
Figure 50: The trapezoidal rule approximates the area under $f(x)$ by a trapezoid.

9.2 Simpson’s Rule:

Unlike the trapezoidal case which uses linear polynomials, the Simpson Rule is what one obtains from using second-order cardinal functions in Lagrange polynomial approximations (quadratic functions).

Schematically, the approximation of the (signed) area under the function $f(x)$ over a finite interval defined by three knots is shown in Figure 51. Here we will choose the panel to be of size $h$ too, but we’ll need to fit 3 quadrature points over $h$, at the two ends of the domain, and somewhere in between.

Let $h = \frac{b - a}{2}$, $x_0 = a$, $x_1 = a + h$, $x_2 = b$, then, integrating the Lagrange polynomial interpolation of $f(x)$ we obtain

\[
\int_a^b f(x)dx = \int_{x_0}^{x_2} \left[ \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \right] dx
\]
Figure 51: The Simpson rule approximates the area under $f(x)$ by a box bounded above by a parabolic function.

\[
\int x_0 \int (x_0 - x_1) f(x_2) \bigg] dx + \\
\int x_0^2 \frac{1}{6} (x - x_0)(x - x_1)(x - x_2)f^{(3)}(\xi(x))dx.
\]

Exercise Find the error for the Simpson Rule. Apply the same strategy used in the estimate of the error for the trapezoidal case. You should get $O(h^4)$ error involving $f^{(3)}$ (the third derivative of $f(x)$ evaluated at some value $\xi \in (x_0, x_2)$).

\[
f(x) = f(x_1) + f'(x_1)(x-x_1) + \frac{f''(x_1)}{2}(x-x_1)^2 + \frac{f'''(x_1)}{6}(x-x_1)^3 + \frac{f^{(iv)}(\xi(x))}{24}(x-x_1)^4
\]
and integrating both sides of this equation it follows that

\[(50) \quad \int_{x_0}^{x_2} f(x) \, dx = f(x_1)(x_2 - x_0) + \]

\[\left[ \frac{f'(x_1)}{2} (x - x_1)^2 + \frac{f''(x_1)}{12} (x - x_1)^3 + \frac{f'''(x_1)}{720} (x - x_1)^4 \right]_{x_0}^{x_2} + \]

\[\frac{1}{24} \int_{x_0}^{x_2} f^{(iv)}(\xi(x))(x-x_1)^4 \, dx.\]

In the error term we obtain \((x - x_1)^4\). Notice that \((x - x_1)^4\) is nonnegative over the interval so we can apply the Weighted Mean Value Theorem

\[\frac{1}{24} \int_{x_0}^{x_2} f^{(iv)}(\xi(x))(x-x_1)^4 \, dx = \frac{f^{(iv)}(\xi_1)}{120} (x-x_1)^5 \bigg|_{x_0}^{x_2}\]

for some \(\xi_1 \in (x_0, x_2)\). Since the points are equally spaced, i.e. \(h = x_2 - x_1 = x_1 - x_0\), we can rewrite (50) as

\[\int_{x_0}^{x_2} f(x) \, dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{12} \frac{1}{3} f^{(iv)}(\xi_2) - \frac{1}{5} f^{(iv)}(\xi_1),\]

(work through this calculation to prove this to yourself). The resulting approximation of the integral is

\[\int_{x_0}^{x_2} f(x) \, dx = 2hf(x_1) + \frac{h^3}{3} f''(x_1) + \frac{h^5}{60} f^{(iv)}(\xi_1).\]

We can approximate the second derivative term using the three point central difference formula

\[f''(x_1) = \frac{f(x_2) - 2f(x_1) + f(x_0)}{h^2} + \frac{h^2}{12} f^{(iv)}(\xi_2),\]

which gives us the result

\[\int_{x_0}^{x_2} f(x) \, dx = \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{36} f^{(iv)}(\xi_2) + \frac{h^5}{60} f^{(iv)}(\xi_1).\]
Applying the Weighted Mean Value Theorem once more yields

\[
\text{Simpson’s rule } \int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90}f^{(iv)}(\xi)
\]

Simpson’s rule is exact for polynomials of degree 3 or less. This is one degree higher than we would have expected based on the Lagrange polynomials. It is a consequence of the symmetry of the grid points which leads to the cancellation of what otherwise would have been the leading error term.

9.3 Discussion

Two important remarks are in order:

1. Both the Trapezoidal and Simpson Rules are examples of what are known as \textit{Newton-Cotes Quadrature Formulas}. These are quadrature rules that use equidistant grid points. Higher order Newton-Cotes formulas are similarly derived, however, when the function to be integrated is non-singular and non-highly oscillatory these low order methods (nothing low about Simpson anyway!) are quite adequate for nonsingular functions. As a matter of fact, Newton-Cotes formulas of degree higher than 8 can lead to unusual results (the weights or coefficients are negative, and numerical sensitivity ensues from the requirement of very delicate cancelations in the final sum).

2. Three simple ways of improving the accuracy of the quadrature rules just discussed are:

   - Use different sized panels (see9.5). A common strategy to decide how big to make each panel is to use derivative information on \( f(x) \) over the interval \([a, b]\) (for this you only need an estimate of the derivative(s), not the actual thing(s)). If the derivative is high, the panels are made smaller.

   - Use unequally spaced points. These are chosen to minimize the truncation error (as we did in Chebishev polynomial interpolation). A common strategy is called \textit{Gaussian Quadrature}, which we cover briefly next (section 9.4).
9.4 Gaussian Quadrature

In this strategy we want to choose the quadrature points so that evaluation of the sum is in some sense optimal. This invariably requires us to use unequally spaced points:

Choose $x_0, x_1 \cdots x_n$ such that the coefficients $c_1, c_2 \cdots c_n$.

\[
\left\| \int_a^b f(x)dx - \sum_{i=1}^n c_i f(x_i) \right\| < \varepsilon,
\]

where $\varepsilon > 0$ is the error we are willing to live with, in some appropriate norm $\| \cdot \|$. The quadrature points $x_i \in [a, b]$ may not be necessarily equally-spaced.

We need to fix the value of $2n$ quantities: the nodes $x_i$ and the weights $c_i$.

In what follows we will denote as $\mathcal{P}_n$ the class of polynomials of degree at most $n$.

Note that polynomials in $\mathcal{P}_{2n-1}(x)$ contain at most $2n$ parameters. We’ll use the “method of undetermined coefficients”, as is done in the following example:

**Example** Let $[a, b] = [-1, 1]$ and $n = 2$, then $c_1, c_2, x_1, x_2$ are chosen so that

\[
\int_{-1}^1 f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)
\]

gives exact result whenever $f(x) \in \mathcal{P}_{2n-1} = \mathcal{P}_3$

i.e. $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$.

Since

\[
\int (a_0 + a_1 x + a_2 x^2 + a_3 x^3)dx = a_0 \int dx + a_1 \int xdx + a_2 \int x^2dx + a_3 \int x^3dx,
\]

we can reduce the problem of choosing the $c_i$ and $x_i$ is equivalent to choosing them so that when $f(x) = 1, x, x^2, x^3$ (a basis of $\mathcal{P}_3$, we get no quadrature error for polynomials of degree at most 3.
These are the necessary conditions:
\[
c_1 + c_2 = \int_{-1}^{1} 1 \, dx = 2 \quad c_1 x_1 + c_2 x_2 = \int_{-1}^{1} x \, dx = 0
\]
\[
c_1 x_1^2 + c_2 x_2^2 = \int_{-1}^{1} x^2 \, dx = \frac{2}{3} \quad c_1 x_1^3 + c_2 x_2^3 = \int_{-1}^{1} x^3 \, dx = 0.
\]
Solving for the unknowns,
\[
c_1 = 1, \quad c_2 = 1, \quad x_1 = \frac{\sqrt{3}}{3}, \quad x_2 = \frac{\sqrt{3}}{3}.
\]
Therefore
\[
\int_{-1}^{1} f(x) \, dx \approx f \left( \frac{-\sqrt{3}}{3} \right) + f \left( \frac{\sqrt{3}}{3} \right),
\]
which gives exact results when \( f \in P^3 \).

More general case: Use orthogonal polynomials on the interval of interest: use Legendre Polynomials \( P_n(x) \) on \([-1, 1]\), Hermite on \((0, \infty)\), Fourier sine and cosine for \([0, 2\pi]\) periodic \(L_2\) functions, use Chebyshev polynomials on \([-1, 1]\), etc.

### 9.4.1 Using Legendre Polynomials on \([-1, 1]\)

Properties of Legendre Polynomials

1. For each \( n \), \( P_n \) is a polynomial of degree \( n \).
2. If \( p(x) \) is a polynomial of degree less than \( n \),
\[
\int_{-1}^{1} p(x)P_n(x) \, dx = 0.
\]
3. They are orthogonal with respect to the weight 1:
\[
\int_{-1}^{1} P_n(x)P_m(x) \, dx = \frac{2}{2n + 1} \delta_{n,m}.
\]
The first few Legendre polynomials are:

\[ P_0(x) = 1 \quad P_1(x) = x \quad P_2(x) = x^2 - \frac{1}{3} \quad P_3(x) = x^3 - \frac{3}{5}x, \quad P_4(x) = x^4 - \frac{6}{7}x^2 + \frac{3}{35}. \]

Note that the roots of these polynomials are distinct and lie in \((-1, 1)\). \(P_n\)'s have symmetry about the origin (they will be even or odd depending on the degree \(n\) of the polynomial).

**Theorem:** Suppose that \(x_1, x_2, \cdots, x_n\) are the roots of the \(n^{th}\) Legendre polynomial \(P_n\) and that for each \(i = 1, 2, \cdots, n\)

\[
c_i = \int_{-1}^{1} \prod_{j=1, j \neq i}^{n} \frac{(x - x_j)(x_i - x_j)}{x_i - x_j} \, dx.
\]

If \(p\) is a polynomial of degree less than \(2n\) then

\[
\int_{-1}^{1} p(x) \, dx = \sum_{j=1}^{n} c_j p(x_j).
\]

In this Gaussian Quadrature strategy, the error for the general function \(f(x)\), defined on \(x \in [-1, 1]\), sufficiently continuous, is

\[
E_n(f) = \frac{f^{(2n)}(\xi)}{(2n)!} \int_{-1}^{1} P_n^2(x) \, dx
= \frac{2}{(2n + 1)!} \left[ \frac{2^n (h!)^2}{(2n)!} \right] f^{(2n)}(\xi) \quad -1 < \xi < 1.
\]

**Proof:** Suppose \(R(x)\) is of degree less than \(n\). Rewrite \(R(x)\) as an \((n - 1)\)-Lagrange polynomial with nodes at the roots of the \(P_n\) polynomial. This representation is exact since the error term involves the \(n^{th}\) derivative of \(R\) and the \(n^{th}\) derivative of \(R\) is 0. Hence,

\[
\int_{-1}^{1} R(x) \, dx = \int_{-1}^{1} \left[ \sum_{i=1}^{n} \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} R(x_i) \right] \, dx
= \sum_{i=1}^{n} \left[ \int_{-1}^{1} \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_j - x_j} \, dx \right] R(x_i) = \sum_{i=1}^{n} c_i R(x_i).
\]

This shows this is fine for polynomials of degree less than \(n\).
If a polynomial \( P(x) \) of degree less then \( 2n \) is divided by the \( n \)th Legendre polynomial \( P_n(x) \) we get

\[
P(x) = Q(x)P_n(x) + R(x),
\]

where \( Q(x) \) and \( R(x) \) are polynomials of degree \( < n \).

Note that the first of the properties of the Legendre polynomials (orthogonality) guarantees that

\[
\int_{-1}^{1} Q(x)P_n(x)dx = 0.
\]

Also, since \( x_i \) is a root of \( P_n \) for each \( i = 1, 2, \ldots, n = 0 \),

\[
P(x_i) = Q(x_i)P_n(x_i) + R(x_i) = R(x_i).
\]

Finally, since \( R(x) \) is polynomial of degree less than \( n \), it follows that

\[
\int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} c_iR(x_i),
\]

so

\[
\int_{-1}^{1} P(x)dx = \int_{-1}^{1} [Q(x)P_n(x) + R(x)] dx = \int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} c_iR(x_i) = \sum_{i=1}^{n} c_iP(x_i).
\]

To find the values of the roots of \( P_n \) consult the web, Abramowitz and Stegun, Stroud and Secrest or symbolic math programs like Maple or Mathematica for Gaussian quadrature.

How do we use this quadrature rule on a finite interval \([a, b]\)? A change of variables converts the problem as defined over the interval \([a, b]\) into one defined over \([-1, 1]\): Let

\[
t = \frac{1}{b - a}(2x - a - b) \text{ then } [a, b] \rightarrow [-1, 1]
\]

and

\[
\int_{a}^{b} f(x)dx = \int_{-1}^{1} f \left( \frac{(b-a)t + b + a}{2} \right) \frac{(b-a)}{2} dt.
\]
9.5 Composite Numerical Integration

Newton-Cotes formulas are unsuitable for large integration intervals. Instead, we can use the idea of approximating the integrand with piece-wise polynomial functions. We split up the integral domain into $m$ equal parts, as in Figure 52.

$$
\int_{a}^{b} f(x) \, dx = \int_{X_0=a}^{X_1} f(x) \, dx + \int_{X_1}^{X_2} f(x) \, dx + \int_{X_2}^{X_3} f(x) \, dx + \cdots + \int_{X_{n-1}}^{b} f(x) \, dx
$$

with $X_i = a + i(b - a)/m$ do each one separately. Each integral is then evaluated using a Newton-Cotes formula such as the trapezoid rule or Simpson’s rule. In the case of Simpson’s rule, this involves evaluating $f(x)$ at the midpoint of each subinterval. The points where $x$ must be evaluated for the composite form of Simpson’s rule are $X_j = x_j = a + jh$ with $h = (b - a)/n$ and $n = 2m$. The resulting formula is:

$$
\int_{a}^{b} f(x) \, dx = \frac{h}{6} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots)
$$
Using the weighted mean value theorem it is straightforward to replace the error term with a single term

\[\int_a^b f(x) \, dx = \frac{h}{6} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)) - \frac{nh^5}{2 \cdot 90} f^{(iv)}(\xi),\]

and since \(h = (b - a)/n\) this can be written as

\[\int_a^b f(x) \, dx = \frac{h}{6} (f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)) - \frac{h^4(b - a)}{180} f^{(iv)}(\xi).\]

(52)

Example: Let's look at using the composite method for evaluating the integral

\[\int_0^5 e^x \, dx.\]

We know that the actual value of this integral is \(e^5 - e^0 = 147.4132\). First we will use the trapezoidal rule and look at how the error decreases as we increase the number of intervals, \(n\), we use. Then we will use Simpson's rule for the integration and again look at how the error changes as we increase \(n\).
Figure 53: Plot of the function $f(x) = e^x$ over the interval of integration along with the trapezoidal approximation for $n = 2$.

Figure 53 shows the function plotted against the two trapezoids that the trapezoidal rule would use to approximate the integral for $n = 2$. As you can see this approximation overestimates the area under $e^x$ considerably. The table below gives the error as 69.8. But the table also shows that as we increase the number of intervals the error decreases significantly. We can also see this in the log-log plot of $n$ versus $-$Error$-$ shown in figure 54. Because this error line is a straight line there is a power law relating $n$ to $-$Error$-$. If we look at the table and compare the errors for $n = 10, 100$ and $1000$ we can see that as $n$ increases by ten the error decreases by approximately 100. Thus we can say that

$$Error(n) \approx Cn^2$$

for some constant $C$. See Table 9.5. We can also use Simpson’s rule to estimate the integral. Simpson’s Rule requires an even number of intervals. Figure 55 shows $f(x) = e^x$ and the polynomial used by the Simpson’s rule to approximate $f(x)$ for $n = 2$. This polynomial underestimates $f(x)$ for the first interval and overestimates $f(x)$ for the second interval. The table of errors below shows that the error for $n = 2$, Simpson’s Rule is much less than for the Trapezoidal Rule with the same number of intervals. The table
Figure 54: Log-log plot showing the absolute error of the Trapezoidal rule numerical integration of $\int_0^5 e^x \, dx$ for different numbers of intervals $n$.

Table 1: Approximation to $\int_0^5 e^x \, dx$, for different $n$. Trapezoidal Rule.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Approximate Integral</th>
<th>Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>373.5329</td>
<td>226.12</td>
</tr>
<tr>
<td>2</td>
<td>217.2227</td>
<td>69.810</td>
</tr>
<tr>
<td>5</td>
<td>159.4976</td>
<td>12.084</td>
</tr>
<tr>
<td>10</td>
<td>150.4715</td>
<td>3.0584</td>
</tr>
<tr>
<td>20</td>
<td>148.1801</td>
<td>0.76698</td>
</tr>
<tr>
<td>50</td>
<td>147.5360</td>
<td>0.12282</td>
</tr>
<tr>
<td>100</td>
<td>147.4439</td>
<td>$3.0710 \times 10^{-02}$</td>
</tr>
<tr>
<td>200</td>
<td>147.4208</td>
<td>$7.6777 \times 10^{-03}$</td>
</tr>
<tr>
<td>500</td>
<td>147.4144</td>
<td>$1.2284 \times 10^{-03}$</td>
</tr>
<tr>
<td>1000</td>
<td>147.4135</td>
<td>$3.0711 \times 10^{-04}$</td>
</tr>
</tbody>
</table>
Figure 55: Plot of the function $f(x) = e^x$ over the interval of integration along with the Simpson’s Rule approximation for $n = 2$.

below shows the absolute error for $n$ going from two to two thousand. Figure 56 also shows this graphically on a log-log plot. Again we have a power law relationship between $n$ and $-\text{Error}$—but in this case it is

$$\text{Error}(n) \approx Cn^4$$

for some constant $C$. So not only is the error less for $n = 2$ but the rate that the error decreases as $n$ increases is considerably greater. For $n = 1000$ we have an error of $5.1187 \times 10^{-10}$ compared with an error of $3.0711 \times 10^{-04}$ for the trapezoidal rule. See also Table 9.5.

9.6 Some Adaptive Quadrature Methods

Not considered here, but the general techniques are:

I Method:

use unequally-spaced panels, placing smaller panels where the function changes more rapidly.
Figure 56: Log-log plot showing the absolute error of Simpson’s rule numerical integration of $\int_0^5 e^x \, dx$ for different numbers of intervals $n$.

Table 2: Approximation to $\int_0^5 e^x \, dx$ as a function of $n$. Simpson Rule.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Approximate integral</th>
<th>Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>165.1193</td>
<td>17.706</td>
</tr>
<tr>
<td>4</td>
<td>149.0933</td>
<td>1.6801</td>
</tr>
<tr>
<td>10</td>
<td>147.4629</td>
<td>$4.9701 \times 10^{-02}$</td>
</tr>
<tr>
<td>20</td>
<td>147.4163</td>
<td>$3.1754 \times 10^{-03}$</td>
</tr>
<tr>
<td>40</td>
<td>147.4134</td>
<td>$1.9957 \times 10^{-04}$</td>
</tr>
<tr>
<td>100</td>
<td>147.4132</td>
<td>$5.1170 \times 10^{-06}$</td>
</tr>
<tr>
<td>200</td>
<td>147.4132</td>
<td>$3.1988 \times 10^{-07}$</td>
</tr>
<tr>
<td>400</td>
<td>147.4132</td>
<td>$1.9994 \times 10^{-08}$</td>
</tr>
<tr>
<td>1000</td>
<td>147.4132</td>
<td>$5.1187 \times 10^{-10}$</td>
</tr>
<tr>
<td>2000</td>
<td>147.4132</td>
<td>$3.1974 \times 10^{-11}$</td>
</tr>
</tbody>
</table>
II Method: Use derivative information or any other information about the integral to dictate where more grid points are to be placed for each panel.

III Method: Use more gridding where needed, and use paneling.

IV Method, Romberg Integration: Uses composite Trapezoidal combined with Richardson extrapolation (see 10).

9.7 Monte Carlo Method for Integrals

Monte Carlo methods are a family of methods that rely on probabilistic ideas. The simple example to be presented here is meant to convey a different way to think about computing, more than about giving the method justice or explaining its underlying theoretical underpinning.

We wish to approximate the integral

\[ \int_a^b f(x) \, dx. \]  

\[ \text{(54)} \]

Here \( x \in \mathbb{R}^1 \), but the more common application of this type of approximation method is when \( x \in \mathbb{R}^n \), where \( n \) is greater than 4 or so. It is also capable of approximating definite Riemann Stieltjes integrals, which are of the form

\[ \int_a^b f(x) \, d\mu(x), \]

where \( \mu(x) \) is a function.

Without loss of generality, we will be further assuming that the function \( f(x) \) over the interval \([a, b]\) can be remapped onto the interval \([0, 1]\). Furthermore, we will assume that \( 0 \leq f(x) \leq 1 \), (by a suitable remapping and partitioning). So that the task of approximating (54) is transformed to approximating the integral

\[ I \equiv \int_0^1 f(x) \, dx. \]

Suppose we choose \( N \) pairs of points \((x_i, y_i)\), both drawn from a uniform probability distribution over \([0, 1]\) (see Figure 57).
Figure 57: Domain of \((x, y)\) pairs. Every time you sample from this distribution, you get a random number for \(x\) and \(y\) in the range \([0, 1]\). The chances of getting a specific number between \([0, 1]\) is the same as any other number in that range.
Figure 58: The value of $z_i$ is 0 for $y_i > f(x_i)$, and 1 if in the shaded region.

Define

$$z_i = \begin{cases} 
0 & \text{for } y_i > f(x_i) \\
1 & \text{for } y_i \leq f(x_i).
\end{cases}$$

See Figure 58. Then, if $n = \sum_i z_i$, we have $n/N \approx I$, or more precisely,

$$I = \frac{n}{N} + \mathcal{O}(1/\sqrt{N}).$$

You can also give the integral a different interpretation: $I$ can be seen as the mean value of the probability density function $f(\xi)$, with $\xi$ from a uniformly distributed density function. Then, the estimate of the mean value is

$$I \approx \frac{1}{N} \sum_{i=1}^{N} f(\xi_i).$$

The basic theorem for Monte Carlo estimates of the integral in $\mathbb{R}^N$ space of $f$ says that

$$\int_V f(y_1, y_2, \ldots, y_n)dy_1dy_2\ldots dy_n \approx V\langle f \rangle \pm \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}},$$

(55)
where

\[ \langle f \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f(y_i) \]

\[ \langle f^2 \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f^2(y_i) \]

so we recognize the last term on the right hand side of (55) as the volume \( V \) times the standard deviation.

This is not a rigorous bound and it assumes that the error is Gaussian. The important thing about (55) is that it says that the integral converges, but very slowly: \( \mathcal{O}(1/\sqrt{N}) \), where \( N \) is the number of \((x_i, y_i)\) pairs. So it is typically impractical, i.e. computationally expensive for integrals in one or two dimensions. However, it is competitive for higher dimensional integrals. In particular, there are many applications in estimation theory in which the integral is a Feynman path integral, which is an integral over ALL possible paths.

The computation of integrals via Monte Carlo depend critically on how many samples you use. But on finite precision machines, it also depends on the quality of the random number generator used. Most machines have, at best, a quasi-random number generator (it looks random, but it just has a really long repeat cycle).

10 RICHARDSON EXTRAPOLATION

It is a general strategy to generate high accuracy by using low-order formulas. You pay for this higher order accuracy by having to compute the problem several times, using different computational parameters, such as a step size parameter \( h \), or the dimension of the computation \( n \). This might not sound all that attractive, but suppose you have written a code with 100 thousand lines, which achieves a certain truncation error. If you wanted to change the truncation error by an order of magnitude or more, you’d probably have to code a new program, as complex and or even more complex than the other one. So, this procedure allows you to use the original code to get higher
order accuracy w/o having to write a whole new code; just run the existing code a few more times.

Naturally, at the end of the day you need to determine whether running a code a number of times is less time consuming than writing a whole new code. Part of scientific computing is to generate a cost-benefit analysis of the strategy used to solve a problem.

The idea is to use low order formulas for which the expression of the truncation error is well known. Then, at the expense of higher computation, one can derive higher order accuracy from the low order formulas. It can be applied when the approximation generates a predictable error that depends on a parameter, such as step size $h$.

To illustrate the procedure, assume we have an approximation $N(h)$ to some quantity $M$. This approximation has an order $h$ truncation error. We need to know what the explicit expression is for the first few terms in the truncation error in order to proceed.

\[ M = N(h) + K_1 h + K_2 h^2 + K_3 h^3 \cdots \]

\( K \)'s are known constants. Here $N(h)$ is an approximation to $M$, to $O(h)$.

Assume $h > 0$ and can be arbitrarily chosen such that $h \to 0$ a better approximation is obtained.

**Aim:** Use extrapolation to remove $O(h)$ error to obtain higher order approximation.

Consider (56) when $\frac{h}{2}$

\[ M = N \left( \frac{h}{2} \right) + K_1 \frac{h}{2} + K_2 \frac{h^2}{4} + K_3 \frac{h^3}{8} + \cdots \]  

Multiply (57) by 2 and subtract (56) to obtain

\[ M = \left[ N \left( \frac{h}{2} \right) + \left( N \left( \frac{h}{2} \right) - N(h) \right) \right] + K_2 \left( \frac{h^2}{2} - h^2 \right) + K_3 \left( \frac{h^3}{4} - h^3 \right) + \cdots \]

---

8 This section is lifted from Kinkaid and Cheney’s book on numerical analysis. It is the most concise and straightforward presentation of the general procedure I know.

9 $\varphi$ in Kincaid and Chenney.
eliminates the $K_1$ term.

\[
\text{let } N_1(h) \equiv N(h) \text{ and } N_2(h) = N_1 \left( \frac{h}{2} \right) + \left[ N_1 \left( \frac{h}{2} \right) - N_1(h) \right].
\]

So, to $O(h^2)$

\[
M = N_2(h) - \frac{K_2}{2} h^2 - \frac{3K_2}{4} h^3 + \cdots \tag{58}
\]

Now, replace $h$ by $\frac{h}{2}$

\[
M = N_2 \left( \frac{h}{2} \right) - \frac{K_2}{8} h^2 - \frac{3K_3}{32} h^3 \cdots \tag{59}
\]

Subtract 4 times (59) minus (58):

\[
3M = 4N_2 \left( \frac{h}{2} \right) - N_2(h) + \frac{3K_3}{4} \left( -\frac{h^3}{2} + h^3 \right) + \cdots
\]

So $M = \left[ N_2 \left( \frac{h}{2} \right) - \frac{N_2(h/2) - N_2(h)}{3} \right] + \frac{K_3}{8} h^3 \cdots$

No we have $O(h^3)$ approximation:

\[
\text{let } N_3(h) \equiv N_2 \left( \frac{h}{2} \right) + \frac{N_2(h/2) - N_2(h)}{3}
\]

\[
M = N_3(h) + \frac{K_3}{8} h^3 + \cdots
\]

Process is continued by constructing the $O(h^4)$ approximation

\[
N_4(h) = N_3 \left( \frac{h}{2} \right) + \frac{N_3(h/2) - N_3(h)}{7}
\]

Then $O(h^5)$ is

\[
N_5(h) = N_4 \left( \frac{h}{2} \right) + \frac{N_4(h/2) - N_4(h)}{15} \text{ and so on}.
\]

Thus, if $M = N(h) + \sum_{j=1}^{m-1} K_j h^j + O(h^m)$

then for each $j = 2, 3, \ldots, m$ we have $O(h^j)$ approximation

\[
N_j(h) = N_{j-1} \left( \frac{h}{2} \right) + \frac{N_{j-1}(h/2) - N_{j-1}(h)}{2^{j-1} - 1}
\]
When can it be used?

When truncation error is the form \[ \sum_{j=1}^{m-1} K_j h^\alpha_j + O(h^\alpha_m) \alpha_1 < \alpha_2 \cdots < \alpha_m. \]

note that the \( K' \)'s are known.

Example

\[ f'(x_0) = \frac{1}{2h} [f(x_0 + h) - f(x_0 - h)] - \frac{h^2}{6} f''(x_0) - \frac{h^4}{120} f^{(4)}(x_0) \cdots \]

formula contains only even powers of \( h \).

\[ O(h^2) : \quad N_1(h) \equiv N(h) = \frac{1}{2h} [f(x_0 + h) - f(x_0 - h)] \]

the \( O(h^{2j}) \)

\[ N_j(h) = N_{j-1} \left( \frac{h}{2} \right) + \frac{N_{j-1}(h/2) - N_{j-1}(h)}{4^{j-1} - 1}, \]

\[ \quad \because \text{since only even powers} \]

where \( j = 2, 3, 4 \cdots \)

Since \( \left( \frac{h}{2} \right)^2 = \frac{h^2}{4} \) the multipliers used to eliminate powers of \( h^2 \)

are not powers of 2 but of 4.

In summary, higher-order approximations are derived with minimal computational cost. However, as \( k \) increases in \( N_1(h/2^k) \) we get more round-off error. Note that there is great potential of loss of precision since you are subtracting nearby numbers from each other.

Exercise Use technique to derive 3, −5, −9 point formulas for derivatives, using the two-point formulas.
11 LINEAR ALGEBRA REVIEW

11.1 Vector Norms

Need a notion of how “close” is \( \{ x^{(k)} \}_{k=1}^{\infty} \) to some \( x \).

**Definition:** Euclidean length \( |x| \equiv \sqrt{(x_1)^2 + (x_2)^2 \cdots (x_n)^2} \), for \( x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \)

Let \( \mathbb{R}^n \equiv \) set all \( n \)-dimensional vectors with real components

\[
R^n = \left\{ x | x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_1, x_2, \ldots, x_n \text{ real} \right\}
\]

A norm in \( \mathbb{R}^n \) is a real-valued function \( \| \cdot \| \) defined on \( \mathbb{R}^n \) satisfying

- \( \|x\| \geq 0 \), and \( \|x\| = 0 \) if and only if \( x = 0 \)
- \( \|\alpha x\| = |\alpha| \|x\| \) \( \forall \alpha \) scalars and \( x \) vectors \( \in \mathbb{R}^n \).
- \( \|x + y\| \leq \|x\| + \|y\| \) \( \forall x \) and \( y \in \mathbb{R}^n \)

Note: A useful variation \( \|x - y\| \geq \|x\| - \|y\| \)

Three \( l_p \) - norms for \( \mathbb{R}^n, p = 1, 2, \infty \), are:

- \( \|x\|_1 = |x_1| + |x_2| + \cdots + |x_n| \)
- \( \|x\|_2 = \sqrt{(x_1)^2 + (x_2)^2 \cdots (x_n)^2} \)
- \( \|x\|_\infty = \max \{ |x_1|, |x_2|, \cdots |x_n| \} \)

**Example**
Take

\( x = [1 - 2 - 2]^T \) then \( \|x\|_1 = 5 \quad \|x\|_2 = 3 \)

and \( \|x\|_\infty = 2 \)
Remark:

With these norms, the concept of relative and absolute error are thus defined

\[ ||e||_p^A = ||x - \hat{x}||_p \quad \text{and} \quad ||e||_p^R = \frac{||x - \hat{x}||_p}{||x||_p} \]

\[ p = 1, 2, \infty. \]

Example Consider the subspace \( \{ x : x \in \mathbb{R}^2, ||x|| \leq 1 \} \)

The \( L_2 \)-norm \( ||x||_2 \leq 1 \) defines a unit ball that in the 2-d plane is a circle. The \( L_1 \)-norm \( ||x||_1 \leq 1 \) defines a “unit ball” that in the 2-d plane is a rhombus. The \( L_\infty \)-norm \( ||x||_\infty \leq 1 \) defines a “unit ball” that in the 2-d plane is a square.

As we saw in Section 6, \( ||x||_1 \geq ||x||_2 \geq ||x||_\infty. \)

Also \( ||x||_1 \leq \sqrt{n}||x||_2 \), \( ||x||_2 \leq \sqrt{n}||x||_\infty \) and \( ||x||_1 \leq n||x||_\infty \)

11.1.1 Matrix Norms

let \( M_n \) denote \((n \times n)\) matrices, \( 0 \) is the \( n \times n \) zero matrix. Then the matrix norm for \( M_n \) is a real-valued function \( || \cdot || \) which is defined on \( M_n \) if \( A \in M_n \) and \( B \in M_n \)

\[ ||A|| \geq 0 \text{ and } ||A|| = 0 \text{ if and only if } A = 0 \]

\[ ||\alpha A|| = ||\alpha|| \cdot ||A|| \text{ for any scalar } \alpha \]

\[ ||A + B|| \leq ||A|| + ||B|| \]

\[ ||AB|| \leq ||A|| \cdot ||B|| \]

The latter two are the triangle inequalities for both add and multiply.

There are many ways to define matrix norms. Concentrate on ones that are compatible with the vector norms

\[ ||A||_1 = \max_{1 \leq j \leq n} \left[ \sum_{i=1}^{n} |a_{ij}| \right] \text{ max column sum} \]

\[ ||A||_a = \max_{1 \leq i \leq n} \left[ \sum_{j=1}^{n} |a_{ij}| \right] \text{ max row sum} \]
\[ ||A||_E \equiv \sqrt{\sum_i \sum_j (a_{ij})^2} \] Frobenius Norm

does not include, since
\[ ||Ax||_1 \leq ||A||_1 ||x||_1, \quad ||Ax||_\infty \leq ||A||_\infty ||x||_\infty, \quad ||Ax||_2 \leq ||A||_E ||x||_2 \]

**Example** Suppose \( Ax = b \) \( x \) is an approximation of \( x \) the exact solution.

let \( r = Ax_c - b \) be residual vector

\[
\begin{pmatrix}
\text{then } r &= Ax_c - b = Ax_c - Ax_t \quad \text{or} \\
\quad x_c - x_t &= A^{-1} r \\
\quad ||x_c - x_t||_1 &= ||A^{-1} r||_1 \leq ||A^{-1}||, ||r||_1
\end{pmatrix}
\]

\[ (60) \]

### 11.1.2 Matrix Norms (Optional, a bit more advanced)

We’ll consider finite-dimensional spaces of reals and assume all of this is a review. Let \( V = \mathbb{R}^n \), let \( || \cdot || \) be a function taking \( V \rightarrow \mathbb{R} \) which satisfies the following “norm axioms.”

(a) \( ||x|| \geq 0 \), where \( ||x|| = 0 \) when \( x = 0 \).

(b) \( ||\alpha x|| = |\alpha|||x|| \), here \( \alpha \in \mathbb{R} \)

(c) \( ||x + y|| \leq ||x|| + ||y|| \), The ”Triangle Inequality,”

for some \( x \in V \) and \( y \in V \).

\( || \cdot || \) is known as a “norm” and yields the “length” of \( x \), say, in the space \( V \) in which \( x \) is an element. It is not unique:

For \( x \in V \)

\[
\begin{align*}
||x||_2 &\equiv \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \\
||x||_1 &\equiv |x_1| + |x_2| + \cdots + |x_n| \\
||x||_p &\equiv [|x_1|^p + |x_2|^p + \cdots + |x_n|^p]^\frac{1}{p} \\
||x||_{\max} &\equiv ||x||_\infty = \max_{1 \leq j \leq n} |x_j|
\end{align*}
\]
The last one comes from $\|x\|_p \to \|x\|_{\text{max}}$ as $p \to \infty$.

**Linear Transformations:**

The general linear transformation from $\mathbb{R}^n \to \mathbb{R}^m$ is of the form

$$
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n
\end{pmatrix}
\to
\begin{pmatrix}
A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + \cdots + A_{1n}x_n \\
A_{21}x_1 + \cdots \\
\vdots \\
A_{m1}x_1 + A_{m2}x_2 + A_{m3}x_3 + \cdots + A_{mn}x_n
\end{pmatrix}
= \begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_m
\end{pmatrix},
$$

or

$$Ax = y.$$

$A$ is an $m \times n$ matrix with entries $A_{i,j}$, $(i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n)$.

The set of all $m \times n$ matrices that are vector spaces under element-wise addition and scalar multiplication that are members of $\mathbb{R}^{m \times n}$ space.

**Matrix Multiplication:** if $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ then $AB$ is an $m \times p$ matrix with entries

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

**Matrix Transpose** $A^T : (A^T)_{ij} = A_{ji}$.

**Trace of Square Matrix:** sum of diagonal elements.

$$tr(A) = \sum_{j=1}^{n} A_{jj}, \quad \text{here } A \in \mathbb{R}^{n \times n}.$$

We can define norms on $\mathbb{R}^{m \times n}$, just as we did in $\mathbb{R}^{n}$. For example

$$\|A\|_F := \sqrt{\sum_{j=1}^{n} \sum_{k=1}^{m} A_{ij}^2} \quad \text{"Frobenious" norm, which can be written as}$$

$$\|A\|_F = \sqrt{tr(A^T A)}.$$  

The problem with $\| \cdot \|_F$ is that it has “forgotten” the relation between linear transformations and matrices. To build this back in we define the “induced
or associated” matrix norm on $\mathbb{R}^{m \times n}$ by $\frac{||Ax||_\beta}{||x||_\alpha}$, with $x \neq 0$, and we take its largest value.

It turns out that the quotient is always bounded; which implies that supremum exists and is actually achieved:

$$||A||_{\alpha,p} \equiv \max_{x \neq 0} \frac{||Ax||_\beta}{||x||_\alpha},$$

is the induced norm.

**Remark:** In infinite-dimensional spaces (e.g., Banach spaces), the associated operator norm is

$$\sup_{x \neq 0} \frac{||Ax||_\beta}{||x||_\alpha},$$

even though supremum is not necessarily attained.

Here $x$ an element of the Banach space, $A$ is an operator acting on the Banach space.

Need to check that the norm axioms apply:

The first two follow from the definition. Let’s check the triangle inequality:

Suppose $A, B \in \mathbb{R}^{m \times n}$

$$||A + B||_{\alpha,\beta} \equiv \max_{x \neq 0} \frac{||(A + B)x||_\beta}{||x||_\alpha} = \max_{x \neq 0} \frac{||Ax + Bx||_\beta}{||x||_\alpha}$$

$$\leq \max_{x \neq 0} \frac{||Ax||_\beta + ||Bx||_\beta}{||x||_\alpha} \leq \max_{x \neq 0} \frac{||Ax||_\beta}{||x||_\alpha} + \max_{x \neq 0} \frac{||Bx||_\beta}{||x||_\alpha}$$

$$||A + B||_{\alpha,\beta} \leq ||A||_{\alpha,\beta} + ||B||_{\alpha,\beta}$$

Two other vital properties which prove useful:

$$||Ax||_\beta \leq ||A||_{\alpha,\beta}||x||_\alpha$$

$$||AB||_{\alpha,\gamma} \leq ||A||_{\alpha,\beta}||B||_{\beta,\gamma}$$

if $B \in \mathbb{R}^{n \times p}$, $|| \cdot ||_{\gamma}$ on $\mathbb{R}^p$.
So the vector norms $\| \cdot \|_p$, for $u \in \mathbb{R}^n$:

$$
\|u\|_1 = \sum_{i=1}^{n} |u_i|,
$$

$$
\|u\|_2 = \left( \sum_{i=1}^{n} |u_i|^2 \right)^{\frac{1}{2}},
$$

$$
\|u\|_p = \left( \sum_{i=1}^{n} |u_i|^p \right)^{\frac{1}{p}}, \quad 1 \leq p < \infty,
$$

$$
\|u\|_\infty = \max_{i=1,n} \{|u_i|\}.
$$

**Remark:** To show that $\| \cdot \|_p \to \| \cdot \|_\infty$ as $p \to \infty$ just apply L’Hôpital’s Rule.

**Associated Matrix Norms on** $A \in \mathbb{R}^{n \times n}$:

$$
\|A\|_1 = \max_{j=1,n} \left\{ \sum_{i=1}^{n} |a_{ij}| \right\},
$$

$$
\|A\|_2 = \max \left\{ \sqrt{\lambda} > 0, \text{ where } \lambda \in \sigma(A^T A) \right\},
$$

where $\sigma(\cdot)$ is the eigenvalue spectrum with components $\lambda$.

$$
\|A\|_\infty = \max_{i=1,n} \left\{ \sum_{j=1}^{n} |a_{ij}| \right\}.
$$

**Remark:** To prove $\|A\|_2$ is given in terms of spectrum, square $A$ and use Lagrange multipliers and the definition of the associated norm.

In thinking about matrix norms it is often useful to have a geometric picture. The matrix norm is the maximum of the norm of $\|Ax\|$ when $\|x\| = 1$. We can also (equally well) define it as the maximum of $\|Ax\|$ when $\|x\| \leq 1$. Geometrically, $\|x\| \leq 1$, is a unit disk if we are working with the $L_2$ norm. The matrix 2-norm can be defined as the radius of the smallest circle that contains the image of the unit disk under the map $A$.

Similarly, for the $L_\infty$ norm, $\|x\| \leq 1$ is a square, $S_1$ centered at 0 with sides parallel to the axes of length 2, and the matrix $\infty$-norm is the half the side length.
length of the smallest square centered at 0 with sides parallel to the axes that contains the image of $S_1$.

The 2-norm and $\infty$-norm are illustrated in the figure below for the matrix $A = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix}$.

Let $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^n$

When does $Au = b$ have a unique solution $u \in \mathbb{R}^n$?

**Theorem:** The following statements are equivalent:

1. $\exists! u$ such that $Au = b \iff \det(A) \neq 0$
2. $\iff Au = 0$ has solution $u = 0$
3. $\iff A$ is invertible

if $\| \cdot \|$ is a vector norm in $\mathbb{R}^n$ then

4. $\iff \exists \alpha > 0$ such that $\|Au\| \geq \alpha \|u\| \quad \forall u$

moreover $\|A^{-1}\| \leq \frac{1}{\alpha}$ where

$\| \cdot \|$ is the associated vector norm.

**Proof**

\[
\|A^{-1}\| = \max_{w \neq 0} \left\{ \frac{\|A^{-1}w\|}{\|w\|} | w \in \mathbb{R}^n \right\} = \max_{w \neq 0} \left\{ \frac{\|u\|}{\|Au\|} | u \in \mathbb{R}^n \right\} \leq \frac{\|u\|}{\alpha \|u\|} = \frac{1}{\alpha}.
\]

**A very useful Theorem:**

**Theorem:** If $\| \cdot \|$ is a matrix norm and $B \in \mathbb{R}^{n \times n}$ with $\|B\| < 1$, then $I + B$ is invertible with

\[
\|(I + B)^{-1}\| \leq \frac{1}{1 - \|B\|}.
\]

**Proof:**

$\|(I + B)u\|$ and want to bound from below:
\[ ||(I + B)u|| \geq \alpha ||u||. \]

We write \((I + B)u - Bu = u\)

Then triangle inequality:
\[
\begin{align*}
||u|| &\leq ||(I + B)u|| + ||Bu||, \\
||u|| &\leq ||u|| - ||Bu|| \\
&\geq ||Bu|| \geq (1 - ||B||)||u|| \\
&\geq \alpha ||u||.
\end{align*}
\]

\[ \square \]

Remark: A very nice characteristic of the Frobenius or Euclidean norm is that they have inner products.

Take \(u \in \mathbb{R}^n\), \(A \in \mathbb{R}^{n \times n}\), then
\[ ||u||_2^2 = u^T u. \]

For example:

if \(\exists \alpha > 0\) with \(u^T Au \geq \alpha u^T u\ \forall u\) then \(A\) is invertible with
\[
\begin{align*}
||A^{-1}|| &\leq \frac{1}{\alpha}, \\
\alpha||u||_2^2 &\leq u^T Au, \\
\alpha||u||_2^2 &\leq u^T Au \leq ||u||_2 ||Au||_2 \Rightarrow \alpha||u||_2 \leq ||Au||_2.
\end{align*}
\]

\[ \square \]

Note: if \(A = A^T\) and satisfies \(u^T Au \geq \alpha ||u||_2^2\), \(\alpha > 0, \forall u\), then \(A\) is called “symmetric positive definite”. In particular, if \(A\) and \(B\) are similar
\[ tr(A) = tr(B) \text{ and } \det(A) = \det(B). \]
11.2 Eigenvalues and Eigenvectors

**Definition:** If $A$ is a square matrix

$$p(\lambda) \equiv \det(A - \lambda I)$$ is the associated characteristic polynomial of $A$.

$\lambda$ is a complex number, generally.

If $\lambda$ is a root of $p$ then $(A - \lambda I)x = 0$ has a nontrivial solution. These zeros are called the eigenvalues of $A$, $x$ are the eigenvectors.

If $\lambda$ is an eigenvalue then $Ax = \lambda x$. Hence, $A$ takes $x$ into a scalar multiple of itself. Suppose, for example that $\lambda$ is real. Then

- (a) $\lambda > 1$ scales the vector up.
- (b) $1 > \lambda > 0$ the vector scales down,
- (c) $\lambda < -1$ the vector changes direction and gets scaled.

The set of all eigenvalues $\lambda$ of $A$ shall be denoted as

$$\sigma(A) = \{\lambda_i\}_{i=1}^r \quad r \leq n.$$  

**Definition:** The spectral radius of $A$ is

$$\rho_\sigma(A) \equiv \max_{\lambda \in \sigma(A)} |\lambda|.$$  

**Theorem:** $A \in M_n$, a square matrix of $n$ rows, then

(i) $[\rho(A^T A)]^{\frac{1}{2}} = ||A||_2$. It implies that $||A||_2 \equiv \max_{||x||_2 = 1} ||Ax||_2$.

(ii) $\rho(A) \leq ||A||$, where $|| \cdot ||$ is any induced norm.

**Example:** Calculate the spectral radius of the following matrix:

205
A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix}, A^T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}

A^T A = \begin{bmatrix} 3 & 2 & -1 \\ 2 & 6 & 4 \\ -1 & 4 & 5 \end{bmatrix}.

Then
\[
\det(A^T A - \lambda I) = \det \begin{bmatrix} 3 - \lambda & 2 & -1 \\ 2 & 6 - \lambda & 4 \\ -1 & 4 & 5 - \lambda \end{bmatrix}
\]
\[= -\lambda^3 + 14\lambda^2 - 42\lambda \]

with roots \( \lambda = 0, \lambda = 7 \pm \sqrt{7}. \)

\[
||A||_2 = \sqrt{\rho(A^T A)} = \sqrt{\max(0, 7 - \sqrt{7}, 7 + \sqrt{7})} \approx 3.106.
\]

Often we’ll have occasion to iterate a matrix. It is thus important to determine what happens when you multiply a matrix with itself repeatedly:

**definition:** \( A \in M_n \) is a **convergent matrix** if
\[
\lim_{k \to \infty} (A^k)_{ij} = 0 \text{ for each } i, j = 1, 2, \ldots, n.
\]

**Example**

\[
A = \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix} \Rightarrow A^2 = \begin{bmatrix} \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix}; A^3 = \begin{bmatrix} \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{1}{4} \end{bmatrix}; A^4 = \begin{bmatrix} \frac{1}{16} & 0 \\ \frac{1}{16} & \frac{1}{8} \end{bmatrix};
\]

\[
A^k = \begin{bmatrix} \left(\frac{1}{2}\right)^k & 0 \\ \left(\frac{1}{2}\right)^{k+1} \end{bmatrix}.
\]

Since
\[
\begin{cases}
\lim_{k \to \infty} \left(\frac{1}{2}\right)^k = 0, \\
\lim_{k \to \infty} \frac{k}{2^{k+1}} = 0,
\end{cases}
\]

206
then $A$ is convergent. It is easy to see that $\frac{1}{2}$ is only eigenvalue of $A$, and thus $\rho(A) = \frac{1}{2}$.

**Theorem:** The following statements are equivalent

(i) $A$ is convergent,

(ii) $\lim_{n \to \infty} ||A^n|| = 0$, in any induced norm.

(iii) $\lim_{n \to \infty} ||A^n|| = 0$, $\forall$ induced norms.

(iv) $\rho(A) < 1$,

(v) $\lim_{n \to \infty} A^n x = 0$ $\forall x$.

11.2.1 Matrix Norm and The Spectral Radius

Let’s see how the matrix 2-norm is related to the spectral radius. The definition of the 2-norm of $A$ is

$$||A||_2 = \max_{||x||_2 = 1} ||Ax||_2.$$  

We can write the square of the 2-norm in terms of the dot product, $||Ax||^2 = Ax \cdot Ax = (Ax)^T Ax = x^T A^T Ax$. Thus

$$||A||_2^2 = \max_{||x||_2 = 1} x^T (A^T A) x.$$  

The matrix $B := A^T A$ is a symmetric matrix, and we can take advantage of the fact that for any symmetric matrix, there is an orthonormal basis of eigenvectors (with real eigenvalues), i.e. a basis consisting of unit length eigenvectors that are pairwise perpendicular. Let that basis be the vectors

207
Let \( b_i, i = 1, \ldots, n \). Let \( \lambda_i \) be the corresponding eigenvalues so that \( Bb_i = \lambda_i b_i \). Note that in this case where \( B = A^T A \), the \( \lambda_i \) must be non-negative real numbers, since we can write
\[
Bb_i = A^T \lambda_i b_i = \lambda_i
\]
and also
\[
Bb_i = b_i^T A^T A b_i = (Ab_i)^T Ab_i = \| Ab_i \|^2 \geq 0.
\]
Using this basis, we can compute the square of the 2-norm of \( A \). The set of unit vectors
\[
\{ x \text{ such that } \| x \|_2 = 1 \},
\]
can also be written as
\[
\{ x \text{ such that } x = \sum_{i=1}^n y_i b_i, \text{ with } \sum_{i=1}^n y_i^2 = 1 \}.
\]
This is because the \( b_i \) are orthonormal. When we take the dot product of \( x \) with itself, all of the cross terms are zero, and, since the dot product of \( b_i \) with itself is 1, the sum ends up being the sum of the squares of the coefficients. In the same way, we can write out
\[
x^T (A^T A) x = x^T B x = \sum_{i=1}^n \lambda_i y_i^2.
\]
We can now rewrite the 2-norm squared of \( A \) as
\[
\| A \|_2^2 = \max_{y_i \text{ such that } \sum_{i=1}^n y_i^2 = 1} \| \sum_{i=1}^n \lambda_i y_i^2 \|.
\]
Thus \( \| A \|_2^2 = \max(\lambda_i) = \max(\| \lambda_i \|) = \rho(B) = \rho(A^T A) \), and
\[
\| A \|_2 = \sqrt{\rho(A^T A)}.
\]
In the special case that \( A = A^T \) is symmetric, we see that the eigenvalues of \( B = A^T A = A^2 \) are the squares of the eigenvalues of \( A \), so \( \sqrt{\rho(A^T A)} = \sqrt{\rho(A)^2} = \rho(A) \), and thus we get the relation
\[
\| A \|_2 = \rho(A),
\]
for symmetric matrices.

This last relation is definitely false for a general matrix \( A \). For example, consider
\[
(61) \quad A = \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix}.
\]

208
The eigenvalues of $A$ are both 1, so $\rho(A) = 1$, however,

$$A^TA = \begin{pmatrix} 1 & 1000 \\ 1000 & 100001 \end{pmatrix}$$

has eigenvalues, approximately, 1000001.999999 and 0.000000999998. Thus the spectral radius is 1000001.999999, and the 2-norm of $A$ is approximately 1000.00099999. This is an important example to show how the matrix norm differs from the spectral radius. Geometrically, $A$ is a *skewing transformation*: it doesn’t increase or decrease area (its determinant is 1), but it deforms squares into very stretched out parallelograms, and circles into long thin ellipses.

For the other norms, the relationship between spectral radius and matrix norm is not as explicit, but we always have the inequality

$$\rho(A) \leq ||A||,$$

for any matrix norm. This follows by taking a unit eigenvector $v$ of $A$ with eigenvalue $\lambda$, using the definition of a matrix norm:

$$||A|| = \max_{x \text{ such that } ||x||=1} ||Ax|| \geq ||\lambda v|| = ||\lambda|| ||v|| = |\lambda|.$$ 

Thus the matrix norm is always larger than the maximum of the absolute values of the eigenvalues, *i.e.*, than the spectral radius. The careful reader will notice that there is the possibility that $\lambda$ and $v$ may be complex valued. If we are working with real spaces, this seems to be a problem. The easiest way to deal with this, is to embed the real vector space $\mathbb{R}^n$ in the complex space $\mathbb{C}^n$, and check that all definitions and results still hold. This indeed is the case.

### 11.3 The Condition Number and Error Estimates

The notion of condition number of a matrix will be defined in terms of the matrix norm. The matrix norm measures the maximum amount by which changes in a vector $x$ are “magnified” in the calculation of $Ax$, in fact, that is the definition of the matrix norm of $A$. The condition number $\kappa(A)$, is defined as the product of the norms of $A$ and $A^{-1}$. It measures the maximum
amount by which the relative error in $x$ can be magnified into a relative error in $Ax$. The spectral radius is used in the calculation of the matrix 2-norm, but in general, it is not the same as the matrix norm.

For the skewing matrix example, (61), the condition number (in the 2-norm) is easy to compute as well. Since

$$A^{-1} = \begin{pmatrix} 1 & -1000 \\ 0 & 1 \end{pmatrix},$$

$||A^{-1}||_2 = ||A||_2$, and the condition number is $||A||_2^2 \approx 1000001.999999$.

Let’s consider

$$Ax = b,$$

where $A$ is an $n \times n$. We think of $b$ as the input and $x$ is the output. We are concerned about how the linear operator $A$ responds to perturbations in the input.

**Theorem:** Suppose $A \in M_n$ nonsingular and $x_e$ approx to $x_t$, the exact solution to $Ax = b$, $b \neq 0$. Then for any compatible norms (matrix and vector) we obtain

$$\frac{1}{||A|| ||A^{-1}||} \frac{||Ax_e - b||}{||b||} \leq \frac{||x_e - x_t||}{||x_t||} \leq \frac{||A|| ||A^{-1}||}{{\kappa}(A)} \frac{||Ax_e - b||}{||b||}. \tag{62}$$

**Proof:** Use (60).

**Definition:** The **Condition Number** is $\kappa(A) = ||A|| ||A^{-1}||$. See (62).

Then (62) can be manipulated to obtain

$$\frac{\varepsilon}{\kappa(A)} \leq \frac{||x_e - x_t||}{||x_t||} \leq \varepsilon \kappa(A).$$

- If $\kappa(A) \geq 1$, but the closer $\kappa(A)$ is to 1, the more accurate $\varepsilon$ becomes as a measurement of the error.
- if $\kappa(A) \gg 1$ the relative error may not be small, even if $\varepsilon$ is small.
Thus, when $\kappa(A) \gg 1$ this indicates $Ax = b$ is ill-conditioned, i.e., small changes input data produce large changes in output $x$. Ultimate ill-conditioning: when $A$ is singular (in this case there are infinitely many vectors $u \neq 0$ such that $Au = 0$, and hence $x_t$ and $x_t + u$ are both solutions of $Ax = b$ even when $A$ and $B$ undergo no change at all.

One can show that the connection between the size of $\kappa(A)$ and how close $A$ is to being singular, if $A$ is nonsingular, is

$$\frac{1}{\kappa(A)} = \min \left\{ \frac{||A - B||}{||A||} \text{ such that } B \text{ is singular} \right\},$$

thus, $A$ can be approximated by a singular matrix if and only if $\kappa(A)$ is large.

Remark: How do we find an estimate of $A^{-1}$? We would like an estimate without having to calculate this inverse. It turns out that, short of calculating the inverse, we are left with some approximate ways to infer this information:

Could estimate as follows:

for $x \neq 0$  \quad $x = A^{-1}Ax$

$||x|| \leq ||A^{-1}||||Ax|| \therefore$

$||A^{-1}|| \geq \frac{||x||}{||Ax||}$

so estimate $||A^{-1}||$ by making $||x||/||Ax||$ as large as possible which vector $x$ to choose? it depends on the problem you are working on.

\[\square\]

11.4 Linear Systems (Continued) and Canonical Forms

Canonical Forms

Schur Normal Form: $A \in \mathbb{C}^{n \times m} \Rightarrow \exists$ a unitary matrix $U$ such that

$$T \equiv U^\dagger AU$$

is upper triangular. Note: the $\dagger$ indicates conjugate transpose (which is also known as the 'Hermitian'). The conjugate transpose of a real matrix is simply the transpose.
Since $T$ is upper triangular and $U^\dagger = U^{-1}$, the characteristic equation of $A$ is of the form
\[
f_A(\lambda) = f_T(\lambda) = (\lambda - t_{11})(\lambda - t_{22}) \cdots (\lambda - t_{mm}),
\]
the eigenvalues of $A$ are diagonal elements of $T$. \hfill \Box

Theorem (Principle Axes): Let $A$ be Hermitian, $A \in \mathbb{C}^{n \times m}$. Hermitian means that $A^\dagger = A$. Then $A$ has $n$ REAL eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ not necessarily distinct, and $n$ eigenvectors $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$ that form an orthonormal basis of $\mathbb{C}^n$. If $A$ is real $\Rightarrow u^{(i)}$ are real and form basis of $\mathbb{R}^n$. Finally, $\exists$ a unitary matrix $U$ for which
\[
U^\dagger AU = D = \text{diag} [\lambda_1, \lambda_2, \ldots, \lambda_n]
\]
is a diagonal matrix with diagonal elements $\lambda_1, \lambda_2, \ldots, \lambda_n$. If $A$ is real $\Rightarrow U$ can be taken as orthogonal. \hfill \Box

Jordan Canonical Form:

definition: “Jordan block” is an $n \times n$ matrix
\[
J_n(\lambda) = \begin{bmatrix}
\lambda & 1 & 0 & \cdots & 0 \\
0 & \lambda & 1 & 0 & \\
& & \ddots & \ddots & \ddots \\
& & & \ddots & 1 \\
0 & \cdots & 0 & \cdots & \lambda
\end{bmatrix}, \quad n \geq 1,
\]
where $J_n(\lambda)$ has the single eigenvalue $\lambda$ of multiplicity $n$ and geometric multiplicity $n$.

Theorem (Jordan Canonical Form:) Let $A$ be $n \times n$ matrix $\Rightarrow \exists$ a nonsingular matrix $P$ for which
\[
P^{-1}AP = \begin{bmatrix}
J_{n_1}(\lambda_1) & & & \\
& J_{n_2}(\lambda_2) & & \\
& & \ddots & \\
& & & J_{n_r}(\lambda_r)
\end{bmatrix}
\]
The eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_r \) needn’t be distinct. For \( A \) Hermitian the Principal Axes Theorem implies we must have \( n_1 = n_2 = n_3 = \cdots = n_r = 1 \) for in that case the sum of the geometric multiplicities must be \( n \), the order of matrix \( A \).

\[ P^{-1}AP = D + N \]
\[ D = \text{diag} \left[ \lambda_1, \ldots, \lambda_r \right] \]

with each \( \lambda_i \) appearing \( n_i \) times on the diagonal of \( D \). The matrix \( N \) has all zero entries except for possible ones on the superdiagonal. It is a nilpotent matrix and it satisfies

\[ N^n = 0. \]

Remark:

\[ P^{-1}AP = D + N \]
\[ D = \text{diag} \left[ \lambda_1, \ldots, \lambda_r \right] \]

Theorem: \( A \) is \( n \times n \) matrix. Then for any operator norm

\[ r(\sigma)(A) \leq ||A||. \]

Moreover, if \( \varepsilon > 0 \) is given \( \Rightarrow \) there’s an operator norm denoted \( || \cdot ||_\varepsilon \) for which

\[ ||A||_\varepsilon \leq r(\sigma)(A) + \varepsilon. \]

Corollary: \( A \) as above. \( r(\sigma)(A) < 1 \) if and only if \( ||A|| < 1 \) for some operator norm.

Theorem \( A \) as above. Then \( A^m \) converges to the zero matrix as \( m \to \infty \) if and only if \( r(\sigma)(A) < 1 \).

Theorem (Geometric Series): \( A \) as above. If \( r(\sigma)(A) < 1 \Rightarrow (I - A)^{-1} \) exists and can be expressed as a convergent series

\[ (63) \quad (I - A)^{-1} = I + A + A^2 + \cdots + A^m + \cdots \]
Conversely, if (63) is convergent ⇒ \( r_\sigma(A) < 1 \).

\[ \text{Theorem: } A \text{ is an } n \times n \text{ matrix. If } ||A|| < 1 \Rightarrow (I - A)^{-1} \text{ exists and has the expansion (63). Moreover} \]

\[ ||(I - A)^{-1}|| \leq \frac{1}{1 - ||A||}. \]

\[ \text{Theorem: Let } A \text{ and } B \text{ be } n \times n \text{ matrices. Assume } A \text{ is nonsingular and suppose that} \]

\[ ||A - B|| < \frac{1}{||A^{-1}||}. \]

then \( B \) is also nonsingular and

\[ ||B^{-1}|| \leq \frac{||A^1||}{1 - ||A^{-1}|| ||A - B||}. \]

and

\[ ||A^{-1} - B^{-1}|| \leq \frac{||A^{-1}||^2 ||A - B||}{1 - ||A^{-1}|| ||A - B||}. \]

\[ \]

\[ 11.5 \text{ Eigenvalues and The Canonical Forms of Matrices} \]

Mostly taken from Atkinson’s Numerical Analysis Book.

The statement 

\[ Ax = \lambda x, \]

with \( x \neq 0 \).

The \( \lambda \) are complex or real numbers, \( x \in \mathbb{C}^n, A \in \mathbb{C}^{n \times n} \) is an eigenvalue problem, \( x \) are the eigenvectors,

In fact, \( \lambda \) is an eigenvalue of \( A \) if and only if

\[ \det(A - \lambda I) = 0. \]
Here $I$ is the $n \times n$ identity matrix.

Let $f \equiv \det(A - \lambda I)$, a polynomial of $n^{th}$ degree. $f = 0$ is the characteristic equation. This equation in $\lambda$ has $n$-roots not necessarily all unique. These roots are the eigenvalues.

definition: Let $A$ and $B$ be square matrices of same order. $A$ is “similar” to $B$ if $\exists$ matrix $P$, nonsingular, for which

$$B = P^{-1}AP$$

by extension, notice that this implies

$$A = Q^{-1}BQ$$ where $Q = P^{-1}$.

$P$ is called the change of basis matrix.

Some properties of similar matrices:

a) If $A$ and $B$ are similar, then $f_A(\lambda) = f_B(\lambda)$.

b) If $A$ and $B$ are similar, then $A$ and $B$ have the same eigenvalues.

c) Since $f(\lambda)$ is invariant under similarity transformations of $A$, the coefficients of $f(\lambda)$ are also invariant under such similarity.

### 11.5.1 Location of Eigenvalues:

For $A$, and $n \times n$ matrix, with $A = [a_{ij}] \ i, j = 1, \ldots, n$.

Define

$$r_i := \sum_{j=1}^{n} |a_{ij}| \ i = 1, 2, \ldots, n.$$  

(64)

Let $Z_i$ denote the circle in the complex plane with center $a_{ii}$ and radius $r_i$:

$$Z_i = \{ z \in \mathbb{C} \text{ such that } |z - a_{ii}| \leq r_i \}.$$  

(65)

Theorem (Gerschgorin): $A$ as above. Let $\lambda$ be an eigenvalue of $A$ then $\lambda$ belongs to one of the circles $Z_i$. Moreover if $m$ of the circles form a connected
set $S$, disjoint from the remaining $n - m$ circles, then $S$ contains exactly $m$ of the eigenvalues of $A$, counted according to their multiplicity as roots of the characteristic polynomial of $A$.

Since $A$ and $A^T$ have the same eigenvalues and characteristic polynomial these results are also valid if summation within the column rather than in the row is used defining the radii in (64).

Proof: To get an intuition of how this works, see Figure 59. Let $\lambda$ be an eigenvalue of $A$ and $x$ its corresponding eigenvector. Let $k$ be such that

$$|x_k| = \max_{1 \leq i \leq n} |x_i| = ||x||_\infty.$$
Then, from $Ax = \lambda x$, the $k^{th}$ component is

$$\sum_{j=1}^{n} a_{kj} x_j = \lambda x_k,$$

$$(\lambda - a_{kk})x_k = \sum_{j=1}^{n} a_{kj} x_j,$$

$$|\lambda - a_{kk}| |x_k| \leq \sum_{j=1}^{n} |a_{jk}| |x_j| \leq r_k |x|_{\infty}.$$

Canceling $||x||_{\infty}$ proves first part of theorem.

Now, let $D := \text{diag } [a_{11}, a_{22}, \ldots, a_{nn}]$ and

$$E = A - D.$$ 

For $0 \leq \varepsilon \leq 1$, let

$$A(\varepsilon) = D + \varepsilon E,$$

and $\lambda_i(\varepsilon)$ be the eigenvalues of $A(\varepsilon)$. Note $A(1) = A$. Here the eigenvalues are roots of

$$f_\varepsilon(\lambda) \equiv \det[A(\varepsilon) - \lambda I].$$

Since the coefficients of $f_\varepsilon(\lambda)$ are functions of $\varepsilon$ and since the roots of any polynomial are continuous functions of its coefficients, it follows that $\lambda(\varepsilon)$ are continuous functions of $\varepsilon$. As $\varepsilon$ varies, each $\lambda_i(\varepsilon)$ changes in complex plane, marking a path from $\lambda_i(0)$ to $\lambda_i(1)$. From the first part of the theorem,

$$Z_i(\varepsilon) = \{ z \in \mathbb{C} \text{ such that } |z - a_{ii}| \leq \varepsilon r_i \} \quad i = 1, 2, \ldots, n,$$

with $r_i$ defined as before. (Examples of these circles are shown in Figure 59). The circles decrease as $\varepsilon$ goes from 1 to 0 and $\lambda(\varepsilon)$ remain within them. When $\varepsilon = 0$ the eigenvalues are

$$\lambda_i(0) = a_{ii}.$$ 

By considering the path $\lambda_i(\varepsilon), 0 \leq \varepsilon \leq 1$, it must remain in the component containing $Z_i(1)$ in which it begins at $\varepsilon = 0$. Thus if $m$ of the circles $Z_i(1)$ form a connected set $S \Rightarrow S$ must contain exactly $m$ eigenvalues $\lambda_i(1)$, as it contains the $m$ eigenvalues $\lambda_i(0)$. 

217
Example) Consider

\[
A = \begin{bmatrix}
4 & 1 & 0 \\
1 & 0 & -1 \\
1 & 1 & -4
\end{bmatrix}
\]

Gershgorin’s Theorem shows eigenvalues must be contained in the circles

\begin{equation}
|\lambda - 4| \leq 1 \quad |\lambda| \leq 2 \quad |\lambda + 4| \leq 2.
\end{equation}

Since first circle is disjoint from remaining ones, there must be a single root in the circle. Since the coefficients of

\[f(\lambda) = \det[A - \lambda I]\]

are real then eigenvalues are complex conjugate pairs. This with (67) implies that there’s a real eigenvalue in [3, 5]. The last 2 circles touch at single point \((-2, 0)\). Using same reasoning, the eigenvalues in these 2 circles must be real. By using (66) of \(A(\varepsilon), \varepsilon < 1\), there’s 1 eigenvalue on \([-6, -2]\) and 1 in \([-2, 2]\). Since it’s easily checked that \(\lambda = -2\) is not an eigenvalue \(\Rightarrow A\) has a real eigenvalue in each of the intervals

\([-6, -2), (-2, 2], [3, 5]\)

the true eigenvalues are \(-3.76010, -4.42931, 420303\).

Example) Consider the \(n \times n\) matrix
Since $A$ is koan as a tridiagonal matrix. It is symmetric and thus all its eigenvalues are real. $r_i$ are all 1 or 2 and all centers $a_{ii} = 4$. Thus all eigenvalues lie in interval $[2, 6]$. Since eigenvalues of $A^{-1}$ are reciprocals of $A$ then
\[
\frac{1}{6} \leq \mu \leq \frac{1}{2}
\]
is eigenvalue range for $A^{-1}$. Thus
\[
||A^{-1}||_2 = \rho_\sigma(A^{-1}) \leq \frac{1}{2}, \text{ independent of } n.
\]

11.5.2 Bounds for Perturbed Eigenvalues

We perturb $A$. What effect does this cause to its eigenvalues? This is useful in studying the linear stability of finite dimensional operator/matrix problems.

Assume the Jordan canonical form of $A$ is diagonal:
\[
P^{-1}AP = \text{diag}[\lambda_1, \ldots, \lambda_n] = D,
\]
and $P$ is nonsingular. The columns of $P$ will be eigenvectors of $A$ corresponding to eigenvalues $\lambda_1, \ldots, \lambda_n$. Matrices for which (68) holds occur often in practice, but this is a special case, nonetheless.
Note: For diagonal matrices $G = \text{diag } [d_1, d_2, d_3, \ldots, d_n]$, we can state
\[ \|G\| = \max_{1 \leq i \leq n} |d_i| . \]

Theorem (Bauer-Fike) Let $A$ be as above. Let $E$ be a perturbation of $A$ and $\lambda$ an eigenvalue of $A + E$. We can estimate that
\[ \min_{1 \leq i \leq n} |\lambda - \lambda_i| \leq \|P\| \|P^{-1}\| \|E\| . \]

\[ \text{(69)} \]

Corollary: If $A$ is Hermitian and $C$ a perturbation of $A$, i.e., The new problem is $A + C$, then
\[ \min_{i \leq i \leq n} |\lambda - \lambda_i| \leq ||E||_2 \]
for any eigenvalue $\lambda$ of $A + C$.

Remark: above says that small perturbation of Hermitian matrices lead to equally small perturbations of eigenvalues.

Theorem (Wielandt-Hoffman:) Let $A$ and $E$ be real symmetric $n \times n$ matrices. Let $\hat{A} = A + E$. Let $\lambda_i$ and $\hat{\lambda}_i, i = 1, 2, \ldots, n$ be eigenvalues of $A$ and $\hat{A}$, respectively, arranged in increasing order. Then
\[ \left( \sum_{j=1}^{n} (\lambda_j - \hat{\lambda}_j)^2 \right)^{1/2} \leq F(E) \]
\[ \text{(70)} \]
where $F(E) = \left( \sum_{i,j=1}^{n} |a_{ij}|^2 \right)^{1/2}$, the Frobenius norm.

A computable error bound for symmetric matrices

Let $A$ be a symmetric $n \times n$ matrix. Let $x$ and $\lambda$ be approximated computed eigenvector and eigenvalue. Define the residual
\[ \eta := Ax - \lambda x . \]
Since $A$ is symmetric then there exists a unitary matrix $U$, such that

$$U^\dagger AU = \text{diag}[\lambda_1, \lambda_2, \ldots, \lambda_n] \equiv D.$$ 

Using this fact, one can then estimate that

$$\min_{1 \leq i \leq n} |\lambda - \lambda_i| \leq \frac{\|\eta\|_2}{\|x\|_2}.$$ 

Exercise Demonstrate that this result is correct.

\[\square\]

### 11.5.3 Eigenvalues of the Symmetric Tri-Diagonal Matrix

Tridiagonal matrices are very common and it is often necessary to find its eigenvalues. This can be accomplished analytically:

Let $T$ be $n \times n$ symmetric tridiagonal matrix. To find $f_n = \det(T - \lambda I)$ introduce the sequence

$$f_k(\lambda) = \begin{bmatrix}
\alpha_1 - \lambda & \beta_1 & 0 & \cdots & 0 \\
\beta_1 & \alpha_2 - \lambda & \beta_2 & & \\
& \ddots & \ddots & \ddots & \\
0 & & \ddots & \ddots & \beta_{k-1} \\
0 & \cdots & 0 & \beta_{k-1} & \alpha_k - \lambda
\end{bmatrix}$$

for $k = 1, 2, \ldots n$. Note

$$f_0(\lambda) = 1.$$ 

Then

$$f_k(\lambda) = (\alpha_k - \lambda)f_{k-1}(\lambda) - \beta_{k-1}^2 f_{k-2}(\lambda)$$

for $2 \leq k \leq n$. 

\[\square\]
12 NUMERICAL LINEAR ALGEBRA

On reading this material (2001) I am finding it shockingly unoriginal, as well as disorganized. This whole chapter needs to be overhauled.

We will want to find approximations of $x$, which obey

$$Ax = b,$$

where $A$ is a matrix of numbers, and $b$ is a column known vector.

We will be focusing on square matrices. Consideration of the under/over-determined problem is beyond the scope of these notes, but the books by G. Strang on linear algebra and the wonderful book by Trefethen and Bau on applied numerical algebra are highly recommended starting points (for the problem to be considered as well as for the under/over-determined problem). The book I used when I was learning this material was Golub and VanLoan. It is an old book, but it is surprisingly contemporary (but unnecessarily advanced for a first exposure to the material).

The most important lessons we’ll learn this chapter are:

- the familiar direct methods for solving linear systems are computationally expensive.
- For large enough problems alternative procedures exist: they rely on iteration and wherever possible, on exploiting special structure.
- Being able to estimate the condition number of the algorithm and/or the effect of round-off error on the results is crucial to deciding on a strategy with which to obtain an approximate solution to the linear-algebraic problem, by direct or iterative means.
- Being able to estimate the computational cost of the algorithm is another important aspect of the calculation.
12.1 Direct Methods for Linear Systems: Gaussian Elimination

Let

\[ A = \begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1m} \\
    a_{21} & a_{22} & \cdots & a_{2m} \\
    \vdots &       & \ddots & \vdots \\
    a_{n1} & \cdots &       & a_{nm}
\end{bmatrix}. \]

A is an $n \times m$ matrix, with $n$ rows, and $m$ columns. We also use $A = a_{ij}$ to denote same matrix.

$Ax = b$ and $Bx = d$ are said to be equivalent, if they have the same solution.

Three operations are permitted to simplify a linear system, of the form (71). These are Elementary Operations:

(a) Row multiply by $\gamma \neq 0$, where $\gamma$ is a scalar.

(b) $\gamma \times $ (Row)$_j$ can be added to Row$_i$

(c) Row$_j$ can be exchanged with Row$_i$

Theorem: if $Bx = d$ is obtained from $Ax = b$ by a finite sequence of elementary operations then $Ax = b$ and $Bx = d$ are equivalent.

The Gaussian Elimination process exploits above rules to form a “reduced” or “triangular” equivalent matrix problem, of the form shown in Figure 60.

Let’s see how we can use row operations: Consider the linear system

\[
\begin{align*}
x_1 - x_2 + 2x_3 - x_4 &= -8 \\
2x_1 - 2x_2 + 3x_3 - 3x_4 &= -20 \\
x_1 + x_2 + x_3 &= -2 \\
x_1 - x_2 + 4x_3 + 3x_4 &= 12
\end{align*}
\]

expressed compactly as $Ax = b$. This system can be expressed in the follow-
Denote each equation (or row) by $E_i$, $i = 1, 2, 3, 4$. Here, we’ve underlined the first entry of $E_1$ and we’ll call this element a pivot. The pivot is always a nonzero element. Here, it is the left-most nonzero element. We ask then: are there elementary operations that will produce zeros in all elements below the pivot element?

Using the following elementary operations, $E_2 - 2E_1$, $E_3 - E_1$, and $E_4 - E_1$, the system is converted into the equivalent problem

\[
\begin{pmatrix}
1 & -1 & 2 & -1 & -8 \\
2 & -2 & 3 & -3 & -20 \\
1 & 1 & 1 & 0 & -2 \\
1 & -1 & 4 & 3 & 12
\end{pmatrix}.
\]

The underlined element is the new pivot. The next operation is to exchange $E_2$ and $E_3$, resulting in
Next, take $2E_3 + E_4 = E_4$, yielding
\[
\begin{pmatrix}
1 & -1 & 2 & -1 & -8 \\
0 & 2 & -1 & 1 & 6 \\
0 & 0 & -1 & -1 & -4 \\
0 & 0 & 2 & 4 & 12
\end{pmatrix},
\]
the pivots have been underlined. At this point we have an upper triangular matrix and back substitution, starting with $E_4$ going back up to $E_1$ gives us
\[
\begin{align*}
2x_4 &= 4 & \Rightarrow x_4 &= 2 \\
-x_3 - x_4 &= -4 & x_3 &= 4 - x_4 = 2 \\
2x_2 - x_3 + x_4 &= 6 & \Rightarrow 2x_2 - 2 + 2 &= 6 & \Rightarrow x_2 &= 3 \\
x_1 - x_2 + 2x_3 - x_4 &= 8 & \Rightarrow x_1 - 3 + 4 - 2 &= -8 & \Rightarrow x_1 &= -7.
\end{align*}
\]

The process can be encoded into an algorithm, called Gaussian Elimination (see 3).

As we will see, there are two types of failure in the approximate solution of linear systems, via the Gaussian Elimination process. First, we need to remind ourselves of failures that should happen in the algorithm, simply because the problem being posed is does not have a solution: Here are two examples:

**Examples**

\[
A = \begin{bmatrix}
1 & 1 & 1 & 4 \\
2 & 2 & 1 & 6 \\
1 & 1 & 2 & 6
\end{bmatrix}, \quad B = \begin{bmatrix}
1 & 1 & 1 & 4 \\
2 & 2 & 1 & 4 \\
1 & 1 & 2 & 6
\end{bmatrix}
\]

$a_{11} = 1 \Rightarrow (E_2 - 2E_1) \rightarrow E_2$ and $E_3 - E_1 \rightarrow E_3$

\[
\tilde{A} = \begin{bmatrix}
1 & 1 & 1 & 4 \\
0 & 0 & -1 & -2 \\
0 & 0 & 1 & 2
\end{bmatrix}, \quad \tilde{B} = \begin{bmatrix}
1 & 1 & 1 & 4 \\
0 & 0 & -1 & -4 \\
0 & 0 & 1 & 2
\end{bmatrix}
\]

225
Note: $a_{22} = a_{32} = 0$

For $\tilde{A}$ $x_3 = 2$, $-x_3 = -2$, $x_2 = 2 - x_1$. $\infty$ number of solutions
For $\tilde{B}$ $x_3 = 2$, $x_3 = 4$ In this case, no solution.

So these are two familiar outcomes in ill-posed problems. But other problems with the Gaussian Elimination algorithm are present when computing in finite-precision.

Performance Issues. See (3):

One of the most vexing problems with direct methods is related to the computational requirements of large linear-algebraic problems. Let’s consider performance, as measured in terms of storage requirements and computational expense, as measured by the operation count. Measuring the computational expense in terms of operation count is appropriate here since we are performing the computation as a string of sequential tasks. This would not be the best way to measure the computational expense if the algorithm were constructed and run on parallel processing machines.

Storage:
Since we can store the $m_{ji}$ in the 0 entries $a_{ji}$ (below the diagonal entries, since these are not involved in the computation after pivoting). Hence, only require 2D array of size $(n, n + 1)$.

Operation count:
Multiply/divide will be considered “long operations”, since they take longer to make (actually, divide takes longer than multiply, but let’s assume that they take the same amount of time.

Add/subtract will be considered short operations, since they take a shorter amount of compute time.

Look at Table 3

In the elimination stage,
Table 3: Simple Gaussian Elimination Algorithm. Solves $A\mathbf{x} = \mathbf{b}$, by doing two sweeps: in the first sweep we row reduce the system. The second sweep solves for the components of the vector $\mathbf{x}$ by "back-substitution."

Input: $A := a_{ij}$, $i = 1: n$, $j = 1: n+1$
$A$ is the augmented matrix.
Output: $x_i$, $i = 1: n$, or an error message.

for $i = 1: n-1$
    Find $p$, the smallest integer with $1 \leq p \leq n$
    AND $a_{pk}$ not equal to 0.
    If no such $p$ can be found, output ('No unique solution')
    if $p$ not equal $k$
        Exchange row $E_p$ and $E_i$, if needed.
    end
    for $j = i+1: n$
        $m_{ji} := a_{ji} / a_{ii}$;
        $E_j = E_j - m_{ji} E_i$ \% the elimination stage
    end
    if $a_{nn} = 0$
        output ('no unique solution')
        stop
    end
for $i = n-1: -1: 1$
    $tmp = 0$
    for $j = i+1: n$
        $tmp = a_{ij} x_j / a_{ii}$
    end
    $x_i = [a_{i,n+1} - tmp]$ \%back substitute
end
output $x$, the solution
• \(n-i\) divisions performed. Also, the replacement \(E_j - m_{ji}E_i\) \(\rightarrow E_j\) requires \(m_{ji}\) multiplied by each term \(E_i\). Hence the resulting subtotal is: \((n-i)(n-i+1)\) multiplications.

• After this is completed, each term of the resulting equation is subtracted from the corresponding term \(E_j\). This requires \((n-i)(n-i+1)\) subtractions.

Thus, for each \(i = 1, 2, \ldots, n-1\) the operations required in the elimination stage are:

- Mult/Div \((n-i) + (n-i)(n-i+1) = (n-i)(n-i+2)\),
- Adds/Subt \((n-i)(n-i+1)\).

Recall the following useful facts:

\[
\sum_{j=1}^{n} 1 = n,
\]
\[
\sum_{j=1}^{m} j = \frac{m(m+1)}{2},
\]
\[
\sum_{j=1}^{n} j^2 = \frac{m(m+1)(2m+1)}{6}
\]

Therefore, summing over \(i\):

\[
\text{Mult/Div: } \sum_{i=1}^{n-1} (n-i)(n-i+2) = (n^2 + 2n) \sum_{i=1}^{n-1} 1 - 2(n+1) \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} i^2 = \frac{2n^3 + 3n^2 - 5n}{6}.
\]

\[
\text{Add/Subt: } \sum_{i=1}^{n-1} (n-i)(n-i+1) = \frac{n^3 - n}{3}.
\]

Now we go to the back substitution portion:
• 1 division

• \( n - i \) multiplies and \( n - i - 1 \) adds for each summation term, then 1 subtract and 1 divide.

So the total operation count in the back substitution portion is:

- Mult/Div: \( 1 + \sum_{i=1}^{n-1} [(n - i) + 1] = \frac{n^2 + n}{2} \);

- Add/Sub: \( \sum_{i=1}^{n-1} [(n - i - 1) + 1] = \frac{n^2 - n}{2} \)

The Total Operation count:

\[
\text{Mult/Div: } \frac{2n^3+3n^2-5n}{6} + \frac{n^2+n}{2} = n^2 - n/3 + \frac{n^3}{3}
\]

\[
\text{Add/Sub: } \frac{n^2-n}{3} + \frac{n^2-n}{2} = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}
\]

Therefore this algorithm requires \( O(n^3) \) operations. If \( n \) is small, no big deal, but if \( n \) is large, \( n^3 \) is very large...clearly, direct methods will not be practical in these circumstances. For large problems we’re sometimes better off with an iterative method (see 12.2).

12.1.1 Ill-conditioning and Finite Precision Errors

So far we’ve assumed \( A, x, b \) are represented exactly on a machine. However, what we really might have on a machine is

\[(A + \delta A)x = b + \delta b, \quad \text{instead of } Ax = b.\]

If \( ||\delta A|| \) and \( ||\delta b|| \) are small, for example, \( O(10^{-t}) \), where \( t \) is large, then no problem. There are errors, but these do not propagate due to the iteration.

We say \( A \) is ill-conditioned if the resulting solution \( x \) is higher-than-linearly sensitive to perturbations in \( b \). In this case, no matter how small these perturbations to \( A \) and \( b \) are, the errors will propagate, sometimes becoming huge, especially if the size of the matrix is large:
Theorem: If $A$ is nonsingular and $||\delta A|| < \frac{1}{||A^{-1}||}$, then the solution $\tilde{x}$ to $(A + \delta A)\tilde{x} = b + \delta b$ approximates $Ax = b$ with error

$$\frac{||\tilde{x} - x||}{||x||} \leq \frac{\kappa(A)}{1 - \kappa(A)(||\delta A||/||A||)} \left( ||\delta b||/||b|| + ||\delta A||/||A|| \right).$$

Therefore, if $A$ is well-conditioned $\kappa(A)$ is small and perturbations in $A$ and $b$ produce small changes in $x$ and vice versa.

\[\square\]

Remark: This is a result that is method-independent.

Example

Wilkinson found that for Gaussian Elimination with pivoting in $t$-digit arithmetic: the problem that was really being solved is

$$(A + \delta A)x = b$$

where the perturbation $||\delta A||_\infty \leq f(n)10^{1-t}\max_{i,j} |a_{ij}|$. Furthermore, he found that $f(n) \approx n$ and at worst $f(n) = 1.01(n^3 + 3n^2)$.

\[\square\]

12.1.2 Finite Precision: Pivoting and Scaling

Problem: In finite precision computations ill conditioning and round off can cause serious problems. A simple way to avoid some of these problems is accomplished through Pivoting or Scaling, or both.

Finite Precision Example:

$$
0.0001x + 1.00y = 1.00 \\
1.00x + 1.00y = 2.00.
$$

The true solution, rounded to 5 significant digits, is $x = 1.00010$ and $y = 0.99990$.

However, without rearrangement $-10000y = -10000 \Rightarrow y = 1.00$ and $x = 0.00$. 

230
With rearrangement

\[
\begin{align*}
1.00x + 1.00y &= 2.00 \\
0.0001x + 1.00y &= 1.00.
\end{align*}
\]

We get \( x = 1.00 \quad y = 1.00 \).

Thus, we get a great improvement by a simple rearrangement of the rows.

Analysis of problem:

Another Example:

\[
\begin{align*}
E_1 : 0.003000x_1 + 59.14x_2 &= 59.17 \\
E_2 : 5.291x_1 - 6.130x_2 &= 46.78,
\end{align*}
\]

has the exact solution \( x_1 = 10.00 \quad x_2 = 1.00 \).

Use 4 digit rounding:

\[
\begin{align*}
a_{11} = 0.003000 \text{ is a pivot and is small.} \\
m_{21} = \frac{5.291}{a_{11}} = 1763.6\overline{6} \text{ rounds to 1763.}
\end{align*}
\]

\[
(E_2 - m_{21}E_1) \rightarrow E_2 \text{ gives}
\]

\[
\begin{align*}
0.003000x_1 + 59.14x_2 &= 59.17, \\
-104300x_2 &= -104400.
\end{align*}
\]

instead of precise value

\[
\begin{align*}
0.003000x_1 + 59.14x_2 &= 59.17 \\
-104309.376x_2 &= -104309.376.
\end{align*}
\]

Hence it is the disparity in the magnitude of \( m_{21}b_1 \) and \( b_2 \) that introduces roundoff. Nevertheless, the error has not yet propagated. Backward substitution yields:

\[ x_2 = 1.001 \text{ close to } x_2 = 1.00. \]

However

\[
x_1 \approx \frac{59.17 - (59.14)(1.001)}{0.003000} = -10.00
\]

231
The solution is the point of intersection of the vectors $E_1$ and $E_2$. The predicted approximation, using 4 digit rounding is indicated by the square box.

contains small error of 0.001, multiplied by $\frac{59.121}{0.00300}$.

Geometrically, it is clear what is happening in this example: see Figure 61.

The problem can be seen most clearly as follows:

if $|a_{11}|$ small compared to $|a_{i1}| \Rightarrow \gamma = -a_{i1}/a_{11}$ is very large.

then $(E_i + \gamma E_1) \rightarrow E_i$ gives

$(a_{i2} + \gamma_{12})x_2 + \cdots (a_{in} + \gamma a_{in})x_n = b_i + \gamma b_1$

if $\gamma$ large, it’s like replacing the $E_i$ by a multiple of $E_i$.

If $\gamma$ small, then no problem, although rounding still occurs.

Strategy: Find row index $I$ such that

$$|a_{I1}| = \max_{1 \leq i \leq n} |a_{i1}|$$

and rewrite the system so that $I^{th}$ equation becomes first equation.

Now $-a_{i1}/a_{I1}$ is small.
Process is repeated, going row by row. This is called Partial Pivoting and it involves row exchanges only. Note that we don’t even have to really affect the exchange in the computer memory. You just need to have a map of the exchanges.

Look at the following partial pivoting example:

\[
\begin{bmatrix}
0.003000 & 59.14 & 59.17 \\
5.291 & -6.130 & 46.78
\end{bmatrix}
\]

1: find \( \max\{ |a_{11}|, |a_{21}| \} = 5.291 = |a_{21}| \)

Since \( |a_{21}| > |a_{11}| \) then exchange \( E_2 \) and \( E_1 \)

then proceed as usual, since \( m_{21} = \frac{a_{21}}{a_{11}} = 0.0005670 \) small.

For larger systems look for largest \( |a_{i1}| 1 \leq i \leq n \). Take

\[
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\
a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn} & b_n
\end{bmatrix}
\]

suppose it’s \( a_{21} \) then “exchange row \( E_{I+1} \) with \( E_2 \).” Use \( a_{21} \) for pivot value and perform row reduce.

\[
\begin{bmatrix}
a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\
0 & \times & \cdots & \times & \times \\
0 & \times & \cdots & \times & \times \\
0 & \times & \cdots & \times & \times \\
0 & \times & \cdots & \times & \times
\end{bmatrix}
\]

and Repeat Procedure...

Note: Algorithm does not require actual row exchange in the original matrix. This is an expensive computational operation that is avoided by simply keeping track of the row operations at the expense of an array that contains the sequence of row exchanges...more later in 12.1.5.

There is another procedure called Full Pivoting. The latter involves row and column exchanges, a much more complex procedure. To convince yourself
of how column exchanges add complexity, just write down a $3 \times 3$ linear algebraic problem and try to keep track of the solution.

In full pivoting the first step is to find $|a_{IJ}| = \max_{1 \leq i,j \leq n} |a_{ij}|$. Retain the $I^{th}$ equation and eliminate the coefficient of $x_J$ in the $I^{th}$ equation, $1 \leq i \leq n$, $i \neq I$ by replacing the $i^{th}$ equation by $(-a_{iJ}/a_{IJ}) \times E_I + E_i \rightarrow E_i$.

Pivoting may still fail. To see this, consider the following example:

**Example**

\[
\begin{align*}
E_1 &: 30.00x_1 + 591400x_2 = 591700 \\
E_2 &: 5.291x_1 - 6.130x_2 = 46.78.
\end{align*}
\]

\[
m_{21} = \frac{5.291}{30.00} = 0.1764.
\]

\[
\Rightarrow 30.00x_1 + 591400x_2 \approx 591700,
\]

\[
-104300x_2 \approx -104400,
\]

\[
\Rightarrow x_1 \approx -10.00 \quad x_2 \approx 1.001 \Leftarrow \text{Wrong.}
\]

A rule of thumb: always be on the lookout for entries that are very disparate. The disparities can sometimes be ameliorated using a scaling strategy: There are a number of scaling strategies.

Here’s now **Row Scaling** is done:

Take $Ax = b$

multiply $E_i$ by $r_i$ \hspace{1cm} $i = 1, \ldots n$. \hspace{1cm} \[
A x = b \rightarrow D_1A x = D_1b \quad D_1 = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \end{bmatrix}
\]

\[
\text{Let } B = D_1 A
\]

Choose $r_i$ so that $B = b_{ij}$ satisfies

\[
\max_j |b_{ij}| \approx 1 \quad i = 1, 2, \ldots, n.
\]
Equation (72) is satisfied if \( r_i = 1/s_i \), where

\[
(73) \quad s_i = \max_j |a_{ij}| \quad i = 1, 2 \ldots, n.
\]

Hence, the rescaling is performed by

\[
(74) \quad b_{ij} = a_{ij}/s_j \quad i, j = 1, 2 \ldots, n.
\]

Unfortunately, (74) can produce additional rounding errors. An alternative strategy is called **scaled partial pivoting**: if the choice of pivot elements with scaling is forced to remain the same as with no scaling, then the solution of scaled is the same as the unscaled case.

**Scaled Partial Pivoting:**

1) Before elimination, compute \( s_i \) using (73) for \( i = 1, 2 \ldots, n \).

2) Before calculating the multipliers at \( k^{th} \) step, scan \( k^{th} \) column of \( A \) and determine \( p \) such that

\[
\frac{|a_{pk}|}{s_p} \geq \frac{|a_{lk}|}{s_l} \quad l = k, k + 1, \ldots, n.
\]

3) if \( p \neq k \rightarrow \) interchange \( p \) and \( k \). The matrix is actually not scaled. That is, no computation on the matrix itself is involved. All that is required is to effect a comparison. (This would be an \( O(n^2) \) operation, which we clearly want to avoid).

**Example**

Solve:

\[
\begin{bmatrix}
0.7000 & 1725 & 1739 \\
0.4352 & -5.433 & 3.271
\end{bmatrix}
\]

using 4 digit rounding. The row sizes are \( s_1 = \max(0.7, 1725) = 1725 \quad s_2 = 5.433 \).

Since \( |a_{11}|/s_1 = 0.0004 < |a_{21}|/s_2 = 0.0801 \), exchange rows 1 and row 2:

\[
\begin{bmatrix}
0.432 & -5.433 & 3.271 \\
0.700 & 1725 & 1739
\end{bmatrix}
\]

235
\[ m_2 = \frac{0.700}{0.4352} = 1.608 \Rightarrow \begin{bmatrix} 0.4352 & -5.433 & 3.271 \\ 1734 & 1734 \end{bmatrix} \]

\[ x_2 = 1.000 \quad x_1 = 20.00, \text{ which is good.} \]

Exercise: Solve the system using 4-digit rounding without partial pivoting, to compare.

\[ \]
1) First step in Gaussian consists

\[(E_j - m_{j1}E_1) \rightarrow E_j\] where \(m_{j1} = \frac{a_{j,1}^{(1)}}{a_{1,1}^{(1)}}\)

Get new matrix

\[A^{(2)} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots \\ : & : & a_{33}^{(2)} & \cdots \\ : & : & : & \cdots \\ : & 0 & : & \cdots \end{bmatrix},\]

so

\[A^{(2)} \mathbf{x} = M^{(1)} A \mathbf{x} = M^{(1)} \mathbf{b} = \mathbf{b}^{(2)}.\]

Here,

\[M^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & 0 & 0 & \cdots & 0 \\ -m_{31} & 0 & 1 & 0 & 0 & \cdots & 0 \\ : & 0 & 0 & 1 & 0 & \cdots & 0 \\ : & : & : & : & \cdots & 0 & 0 \\ : & 0 & 0 & 0 & \cdots & 1 & 0 \\ -m_{n1} & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}.\]

Construct \(M^{(2)}\) in same way with \(m_{j2} = \frac{a_{j,2}^{(2)}}{a_{2,2}^{(1)}}\). Then

\[A^{(3)} \mathbf{x} = M^{(2)} A^{(2)} \mathbf{x} = M^{(2)} M^{(1)} A \mathbf{x} = M^{(2)} M^{(1)} \mathbf{b} = \mathbf{b}^{(3)}.\]

Therefore if have \(A^{(k)} \mathbf{x} = \mathbf{b}^{(k)}\) then multiply (left) by

\[M^{(k)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -m_{k+1,k} & 1 & 0 & \cdots & 0 \\ : & : & : & : & \cdots & 0 & 0 \\ : & 0 & -m_{n-1,k} & 0 & \cdots & 1 & 0 \\ 0 & 0 & -m_{n,k} & 0 & \cdots & 0 & 1 \end{bmatrix}.\]
This is the *Gaussian transformation matrix*.

Multiplication by the Gaussian transformation matrix leads to
\[
A^{(k+1)} x = M^{(k)} A^{(k)} x = M^{(k)} b^{(k)} = b^{(k+1)} = M^{(k)} \cdots M^{(1)} b.
\]

The process ends with \( A^{(n)} x = b^{(n)} \) where

\[
A^{(n)} = M^{(n-1)} M^{(n-2)} \cdots M^{(1)} A \quad \text{is upper triangle.}
\]

and \( A^{(n)} = \)
\[
\begin{bmatrix}
  a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1,n}^{(1)} \\
  0 & a_{22}^{(2)} & & \\
  \vdots & \vdots & \ddots & \\
  0 & 0 & a_{n-1,n}^{(n-1)} & \\
  0 & 0 & 0 & a_{nn}^{(n)}
\end{bmatrix} = U.
\]

Now we tackle the \( L \) part:

1) Recall that \( M^{(k)} \) is used to compute
\[
A^{(k+1)} x = M^{(k)} A^{(k)} x = b^{(k+1)},
\]

where \( M^{(k)} \) generates

\[
(E_j - m_{j,k} E_k) \to E_j \quad j = k+1, \ldots, n.
\]

To reverse the effects of the transformation and return to \( A^{(n)} \)

requires \( (E_j + m_{j,k} E_k) \to E_j \) be performed \( j = k+1, \ldots, n. \)

This is equivalent to multiplying by the inverse of \( M^{(k)} \):

\[
L^{(k)} = [M^{(k)}]^{-1} = 
\begin{bmatrix}
  1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
  0 & 1 & 0 & 0 & 0 & \cdots & 0 \\
  0 & 0 & 1 & 0 & 0 & \cdots & 0 \\
  0 & 0 & m_{k+1,k} & 1 & 0 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \ddots & 0 & \cdots & 0 \\
  0 & 0 & m_{n-1,k} & 0 & \cdots & 1 & 0 \\
  0 & 0 & m_{n,k} & 0 & \cdots & 0 & 1
\end{bmatrix}.
\]
Let \( L = L^{(1)} L^{(2)} \cdots L^{(n-1)} \) = 
\[
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
m_{21} & 1 & \cdots & 0 \\
\vdots & 1 & \cdots & \vdots \\
m_{n,1} & \cdots & m_{n,n-1} & 1
\end{bmatrix}
\]

The product of \( L \) with \( U = M^{(n-1)} \cdots M^{(2)} M^{(1)} A \) gives

\[
LU = L^{(1)} L^{(2)} \cdots L^{(n-3)} L^{(n-2)} M^{(n-1)} M^{(n-2)} M^{(n-3)} \cdots M^{(2)} M^{(1)} A
\]

Theorem: If Gaussian elimination can be performed in \( Ax = b \) without row interchange \( \Rightarrow A = LU \) factorization is possible.

Note: \( A = LU \) is not a unique factorization. For example let

\[
A = \begin{pmatrix}
4 & 1 \\
1 & 2
\end{pmatrix} = \begin{pmatrix}
4 & 0 \\
1 & 7/4
\end{pmatrix} \begin{pmatrix}
1 & 1/4 \\
0 & 1
\end{pmatrix} = \begin{pmatrix}
1 & 0 \\
1/4 & 1
\end{pmatrix} \begin{pmatrix}
4 & 1 \\
0 & 7/4
\end{pmatrix}.
\]

Thus, in general, one can move the diagonal factors around. However, the \( LDU \) decomposition is unique. Here, \( D \) is a diagonal matrix, \( L \) and \( U \) are unit lower and upper triangular matrices, respectively.

Suppose that the matrix factors as \( A = LU \) using the Gaussian elimination algorithm algorithm 3 with no row exchanges needed. The diagonal entries of \( L \) are all 1’s with this algorithm, i.e. \( L \) is unit lower triangular. We can further factor \( U = DU' \), taking \( D \) to be a diagonal matrix with \( d_{ii} = u_{ii} \), and \( u'_{ij} = u_{ij}/u_{ii} \). This factorization is unique. Suppose that we have \( A \) nonsingular with \( A = LDU = L'D'U' \). Then \((L')^{-1}LD = D'U'U^{-1}\) The left side of this equation is a product of lower triangular matrices, and is thus lower triangular, the right side is upper triangular, so, their common value is diagonal. Call that value \( D'' \). then we have \((L')^{-1}LD = D'' \), and so \((L')^{-1}L = D^{-1}D'' \) It is easily checked that the product of triangular matrices with unit diagonals is again a triangular matrix with unit diagonal. This shows that \((L')^{-1}L = I = D^{-1}D'' \). Thus \( L' = L \) and \( D = D'' \). Similarly,
we get that \((D')^{-1}D'' = U'U^{-1}\) and so \(U = U'\) and \(D' = D''\). This proves that the \(LDU\) factorization is unique.

\(\square\)

Not every matrix can be \(LU\)-decomposed:

Example:

\[
\begin{align*}
4x_2 &= 8, \\
4x_1 + 2x_2 &= 17.
\end{align*}
\]

\[
A = \begin{bmatrix} 0 & 4 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \ell_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}
\]

\[
u_{11} = 0
\]

\[
\ell_{21}u_{11} = 4 \therefore \text{cannot satisfy.}
\]

\[
A = \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix}.
\]

This ensures that no \(u_{rr}\) is 0. This rearrangement is always possible if the solution is unique.

\(\square\)

12.1.4 The Special LU’s: Doolittle, Crout’s, and Choleski Algorithms

The following are LU decomposition strategies, which apply to special matrices:

Doolittle Algorithm: get \(\ell_{ii} = 1\) \(i = 1 \ldots n\). “Unit lower Triangular”

Crout’s Algorithm: get \(u_{ii} = 1\) \(i = 1 \ldots n\) “Unit Upper Triangular”

Choleski: \(U = L^T\) \(\ell_{ii} = u_{ii}\) \(i=1 \ldots n\)

The last of these is used a lot in symmetric definite matrices.

LDL factorization of symmetric matrices:
One of the important strategies in numerical linear algebra is that whenever possible, we take advantage of any special structure of matrices in our computations. One important class of “special” matrices is the symmetric matrices \( A = A^T \). Obviously, we can store such matrices using roughly half the memory, it is natural to ask if we can solve the system \( Ax = b \) in such a way that we take advantage of the symmetry.

Let’s start by looking at a simple \( 2 \times 2 \) example:

Let \( A = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix} \), then we can start the factorization process as

\[
A = \begin{pmatrix} 1 & 0 \\ 1/4 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 \\ 0 & 7/4 \end{pmatrix}
\]

Notice that the symmetry is lost. We can repair that problem by continuing:

\[
A = \begin{pmatrix} 1 & 0 \\ 1/4 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 \\ 0 & 7/4 \end{pmatrix} \begin{pmatrix} 1 & 1/4 \\ 0 & 1 \end{pmatrix}
\]

This factors the matrix \( A = LDL^T \) as a product of a lower triangular matrix with 1’s on the diagonal, times a diagonal matrix \( D \) times the transpose of the lower triangular matrix. In this case the diagonal entries are positive, so we can factor further as

\[
A = \begin{pmatrix} 1 & 0 \\ 1/4 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 1 & 1/4 \\ 0 & 1 \end{pmatrix}
\]

\[
= \begin{pmatrix} 2 & 0 \\ 1/2 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 2 & 1/2 \\ 0 & \sqrt{7}/2 \end{pmatrix}
\]

That is we factor \( A = LL^T \) with \( L \) a lower triangular matrix.

Before we look at algorithms for the solution of this problem it is worth recalling some basic facts about matrices:

1. \( (AB)^T = B^T A^T \)
2. \( (AB)^{-1} = B^{-1} A^{-1} \)
3. \( (A^{-1})^T = (A^T)^{-1} \)
4. The product of upper (lower) triangular matrices is upper (lower) triangular.

5. If an upper (lower) triangular matrix is invertible, the inverse is also upper (lower) triangular.

If $A$ is symmetric and it is factored as $A = LDU = U^T D L^T$, then by uniqueness, $U = L^T$ and $A = LDL^T$. Now that we know that a symmetric (non-singular) matrix which has an $LU$ factorization can be uniquely factored as $LDL^T$, we can ask how this factorization can be implemented in an algorithm. We could just use the Gaussian elimination algorithm, and extract the $D$ entries after we had produced $U$. This works, but is not taking full advantage of the symmetry. An efficient algorithm avoids storing or calculating the upper triangle of entries at all. The following algorithm is not optimized for storage, but takes advantage of the symmetry to cut the work in half. This algorithm is from section 4.1.2 of Golub and Van Loan’s book on linear algebra:

input: A a symmetric nonsingular matrix with an LU factorization
output: L lower triangular and d, the diagonal entries of D
for j=1:n
  # Note this loop is not executed at all when j=1
  for i=1:j-1
    v(i)=L(j,i)d(i)
  end
  # v is a temporary vector of size n
  # note that the second term (the sum) is 0 if j=1
  v(j)=A(j,j)-∑_{i=1}^{j-1}L(j,i)v(i)
  d(j)=v(j)
  # if this d(j)=v(j) is zero, that means that A had no LU factorization
  # real code should test for this error condition.
  L(j+1:n,j)=(A(j+1:n,j)-∑_{i=1}^{j-1}L(j+1:n,i)v(i))/v(j)
end

In the special case where the entries of $D$ are all positive, it is possible to take the square root of the diagonal matrix $D = D'' D''$, with $d''_{ii} = \sqrt{d_{ii}}$. Then we can define $L' = LD''$ and we have $A = L'(L')^T$. It turns out that
this special case arises in many contexts, and there is a name for this kind of symmetric matrix.

Definition: A symmetric matrix $A = A^T$ is said to be positive definite, if for any vector $x \neq 0$, $x^T A x > 0$.

Theorem A symmetric matrix is positive definite if and only if it has a factorization $A = LL^T$ with $L$ a nonsingular lower triangular matrix.

Proof: If $A = LL^T$ with $L$ nonsingular, then $x^T A x = x^T L L^T x$. Let $u = L^T x$, then we have $x^T A x = u^T u = \sum_{i=1}^{n} u_i^2$, and we have that unless $u = 0$, $x^T A x > 0$. The only way $u = L^T x$ could be 0 is if $x = 0$, since $L$ is nonsingular, so if $x \neq 0$, $x^T A x > 0$.

On the other hand, if we assume that $A$ is positive definite, then write

$$A = \begin{pmatrix} \alpha & v^T \\ v & C \end{pmatrix}$$

since $A$ is positive definite, $e_1^T A e_1 = \alpha > 0$ so we can factor

$$A = \begin{pmatrix} 1 & 0 \\ v/\alpha & I \end{pmatrix} \begin{pmatrix} \alpha & 0 \\ 0 & C - vv^T/\alpha \end{pmatrix} \begin{pmatrix} 1 & v^T/\alpha \\ 0 & I \end{pmatrix}$$

Let $A' = C - vv^T/\alpha$. If we can show that $A'$ is positive definite, we will have shown inductively that the factorization works. Let $u \neq 0 \in \mathbb{R}^{n-1}$. Then define $x \in \mathbb{R}^n$ by $x^T = (-u^T v/\alpha, u^T)$.

$$x^T A x = (-u^T v/\alpha, u^T) \begin{pmatrix} 0 & -(u^T v)/\alpha + Cu \\ -(u^T v)/\alpha + Cu & u^T A' u \end{pmatrix} = u^T A' u,$$

so $u^T A' u > 0$ whenever $U \neq 0$. That is $A'$ is positive definite. Continuing inductively we conclude we can factor $A = LDL^T$ with all diagonal entries $d_{ii} > 0$. As we saw above, this is sufficient to allow us to get the desired factorization.

\[ \square \]

The factorization of a symmetric positive definite matrix as $A = LL^T$ is called Cholesky factorization. The algorithm can be implemented quite efficiently:

input: A, symmetric positive definite
output: A, (L overwrites the lower triangle of A)
for k=1:n-1
# if A(k,k) is not positive, A was not positive definite,
# real code should test for this possibility
   A(k,k)=sqrt(A(k,k))
# this scales the column
   A(k+1:n,k)=A(k+1:n,k)/A(k,k)
# and the inner loop subtracts the rank one matrix
# vv^T/α from the remaining lower block.
   for j=k+1:n
      A(j:n,j)=A(j:n,j)-A(j:n,k)A(j,k)
   end
end

This is algorithm 4.2.2 from Section 4.2.5 of the book by Golub and Van Loan.

Remark: Note that the algorithm never calls for access to any elements above the diagonal. This is possible because the matrix is symmetric; we always access the lower triangle. This frees up memory.

Example Consider the symmetric matrix

\[
A = \begin{bmatrix}
  1 & 1 & 4 & -1 \\
  1 & 5 & 0 & -1 \\
  4 & 0 & 21 & -4 \\
 -1 & -1 & -4 & 10 \\
\end{bmatrix}
\]

In fact we do not need the elements above the diagonal because they will just be the same as below the diagonal by symmetry. All we need is

\[
A = \begin{bmatrix}
  1 & \cdot & \cdot & \cdot \\
  1 & 5 & \cdot & \cdot \\
  4 & 0 & 21 & \cdot \\
 -1 & -1 & -4 & 10 \\
\end{bmatrix}
\]

We will follow the above algorithm to calculate L, the lower triangular matrix.

For k = 1 the steps we need to complete are
• $A_1(1, 1) = \sqrt{A(1, 1)}$
• $A_1(2 : 4, 1) = A(2 : 4, 1)/A(1, 1)$
• $A_1(2 : 4, 2) = A(2 : 4, 2) - A_1(2 : 4, 1)A_1(2, 1)$
• $A_1(3 : 4, 3) = A(3 : 4, 3) - A_1(3 : 4, 1)A_1(3, 1)$
• $A_1(4, 4) = A(4, 4) - A_1(4, 1)A_1(4, 1)$

Remark The above operations should be row operations. However we can exploit the symmetry of the matrix and use the columns instead. This way we do not need to explicitly store the matrix elements from above the diagonal.

This gives us the new matrix $A_1$

$$A_1 = \begin{bmatrix}
1 & & & \\
1 & 4 & & \\
4 & -4 & 5 & \\
-1 & 0 & 0 & 9
\end{bmatrix}.$$

For $k = 2$ we perform the following steps

• $A_2(2, 2) = \sqrt{A_1(2, 2)}$
• $A_2(3 : 4, 2) = A_1(3 : 4, 2)/A_2(2, 2)$
• $A_2(3 : 4, 3) = A_1(3 : 4, 3) - A_2(3 : 4, 2)A_2(3, 2)$
• $A_2(4, 4) = A_1(4, 4) - A_2(4, 2)A_2(4, 2)$

which gives the following matrix

$$A_2 = \begin{bmatrix}
1 & & & \\
1 & 2 & & \\
4 & -2 & 1 & \\
-1 & 0 & 0 & 9
\end{bmatrix}.$$

For $k = 3$ we have

245
• $A_3(3, 3) = \sqrt{A_2(3, 3)}$
• $A_3(4, 3) = A_2(4, 3)/A_3(3, 3)$
• $A_3(4, 4) = A_2(4, 4) - A_3(4, 3)A_3(4, 3)$

In this example the matrix $A_3$ is the same as the matrix $A_2$.

For $k = 4$ we just have

• $A_4(4, 4) = \sqrt{A_3(4, 4)}$

giving

$$A_4 = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ 1 & 2 & \cdot & \cdot \\ 4 & -2 & 1 & \cdot \\ -1 & 0 & 0 & 3 \end{bmatrix}. $$

From this we can see that

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

and

$$LL^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 4 & -2 & 1 & 0 \\ -1 & 0 & 0 & 3 \end{bmatrix}\begin{bmatrix} 1 & 1 & 4 & -1 \\ 1 & 2 & -2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 4 & -1 \\ 1 & 5 & 0 & -1 \\ 4 & 0 & 21 & -4 \\ -1 & -1 & -4 & 10 \end{bmatrix} = A$$

Remark: The reduction of work realized in using the symmetry of our matrices is not the only reason that we employ Cholesky factorization whenever we can. Cholesky factorization has superior stability properties.
12.1.5 Permutation Matrix

We should note that there are a number of matrix types where solution without row interchange is possible. However, row exchange is needed to control round-off error when this sort of error is to be ameliorated. We introduce next the *Permutation Matrix*. It is a very useful tool, for a variety of things, but here we will emphasize its role in elegantly effecting row exchanges.

An \((n \times n)\) matrix can be used to rearrange or permute rows of a given matrix: \(P\) has precisely 1 entry whose value is 1 each column and each row, other entries are zero.

**Example** \(P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\)

To see what this Permutation matrix does to a similarly-sized matrix \(A\), we left-multiply

\[
P A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}
\]

So we achieved no change in row 1, and an exchange of rows 2 and 3. We get different answer for right multiply \(AP\), by the way.

**Properties**

Two are useful with respect to Gaussian elimination:

1) As described above, \(PA\) permutes the rows of \(A\)

2) if \(P\) is a permutation matrix \(\Rightarrow P^{-1}\) exists and it is orthogonal: \(P^{-1} = P^T\).

Remark: We have seen that for any nonsingular matrix \(A \Rightarrow Ax = b\) can be solved by Gaussian elimination, with the possibility of row interchanges, then there is a rearrangement that could result in an equivalent matrix problem that can be solved without row interchanges. That is, for \(A\) non-singular.
there exists an equivalent problem

\[ PAx = Pb \]

which can be solved without row interchanges. Hence,

\[ PA = LU. \]

Since \( P^{-1} = P^T \)

\[ \Rightarrow A = (P^T L)U \]

However, unless \( P = I \) \( P^T L \) is not lower triangular.

\[ \square \]

Example Let’s take

\[ A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}. \]

No \( LU \) decomposition is possible since \( a_{11} = 0 \). However, let’s use the permutation matrices to transform the problem:

\[ E_1 \leftrightarrow E_2, \text{ followed by } E_3 - E_1 \rightarrow E_3, \ E_4 - E_1 \rightarrow E_4 \Rightarrow \]

\[ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & -1 & 0 & 4 \end{bmatrix} \]

then \( E_2 \leftrightarrow E_3, (E_4 + E_2) \rightarrow E_2 \)

\[ U = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \]

then \( (E_4 - E_3) \rightarrow E_3 \)
The permutations associated with $E_1 \leftrightarrow E_2$ and $E_2 \leftrightarrow E_4$ are:

\[ U = \begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 3 \\
\end{bmatrix}. \]

Thus, $P = P_2 P_1$, and $P^{-1} = P^T$.

Now we do Gaussian elimination if we were solving for $Ax = b$ on $PA$:
\[ E_4 - E_1 \rightarrow E_4 \quad (\beta) \]
\[
\begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & -1 & 0 & 4
\end{bmatrix}
\]

\[ E_4 + E_2 \rightarrow E_4 \quad (\gamma) \]
\[
\begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 5
\end{bmatrix}
\]

\[ E_4 - E_3 \rightarrow E_4 \quad (\delta) \]
\[
\begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 4
\end{bmatrix}
= U
\]

(75)

So to write down the \(LU\) factorization of \(PA\):

From (\(\alpha\)) we surmise that

\[
L = \begin{bmatrix}
1 & 1 & 0 & 0
\end{bmatrix}
\]

From (\(\beta\)), (\(\gamma\)) and (\(\delta\)) we see that

\[
L = \begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & -1 & 1 & 1
\end{bmatrix}.
\]

Since rows 1 and 3 are left unchanged

\[
L = \begin{bmatrix}
1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
1 & -1 & 1 & 1
\end{bmatrix}.
\]
Hence, \( PA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} \equiv LU \)

Note that if we multiply \( P^T \Rightarrow A = (P^T L)U \) we recover \( A \).

In summary, Gaussian elimination can be performed on \( PA \) without row interchange using \( E_2 - E_1 \rightarrow E_2, E_4 - E_1 \rightarrow E_4, E_4 + E_2 \rightarrow E_4, E_4 - E_3 \rightarrow E_4 \):

\[
P A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} = LU.\]

Don’t forget to multiply \( b \) by \( P \), before the Gaussian Elimination is effected.

\[\square\]

### 12.1.6 Other Special Matrix Types

What types of Matrices permit Gaussian Elimination without Row interchange? One important class is strictly diagonally dominant matrices.

definition: The \( n \times n \) matrix \( A \) is Strictly Diagonally Dominant if

\[ |a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \text{ holds for } i = 1, 2 \ldots n. \]

Example: Consider 2 unrelated matrices,

\[
A = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{bmatrix},
B = \begin{bmatrix} 6 & 4 & -3 \\ 4 & -2 & 0 \\ -3 & 0 & 1 \end{bmatrix}.
\]
Note: $A$ is non-symmetric, and $B$ is symmetric. $A$ is diagonally dominant, $B$ is not. $A^T$ is not strictly diagonal and neither is $B^T = B$

Let’s recall a fundamental theorem about linear systems:

**Theorem.** The following statements are equivalent for an $n \times n$ matrix $A$:

a) $Ax = 0$ has a unique solution $x = 0$.

b) The linear system $Ax = b$ has unique solution for any $n$-column vector $b$.

c) $A$ is nonsingular, *i.e.*, $A^{-1}$ exists.

d) $\det A \neq 0$.

e) Gaussian elimination with row exchanges can be performed on $Ax = b$.

**Theorem:** A strictly diagonally dominant matrix $A$ is nonsingular. Moreover, Gaussian elimination can be performed without row interchanges on $Ax = b$ to obtain unique solution and computation is stable with regards to growth of round-off errors.

**Proof:** First, prove $A$ is not singular by contradiction. We’ll assume that $A$ is an $n \times n$ matrix, and that the vectors $b$ and $x$ are $n$-dimensional vectors.

Take $Ax = 0$ and suppose that non-zero $x$ with components $x_i$ exists.

Let $k$ be index for which

$$0 < |x_k| = \max_{1 \leq j \leq n} |x_j|$$

Since $\sum_{j=1}^{n} a_{ij}x_j = 0$ for each $i = 1, 2, \ldots, n$, we have,

when $i = k$, $a_{kk}x_k = -\sum_{\substack{j=1\ j \neq k}}^{n} a_{kj}x_j$.  

252
This implies that

\[ |a_{kk}| |x_k| \leq - \sum_{j=1 \atop j \neq k}^{n} |a_{kj}| |x_j|, \]

or

\[ |a_{kk}| \leq \sum_{j=1 \atop j \neq k}^{n} |a_{kj}| \frac{|x_j|}{|x_k|} \leq \sum_{j=1 \atop j \neq k}^{n} |a_{kj}|. \]

This contradicts the strict diagonal dominance of \( A \). Therefore the only solution of \( Ax = 0 \) is \( x = 0 \).

Next, we prove that Gaussian elimination can be performed without row exchanges: We need to show that the matrices \( A^{(2)} \ldots A^{(n-1)} \), generated by Gaussian process, are strictly diagonal dominant: Recall that a symmetric matrix \( A \) is positive definite if \( x^T A x > 0 \), for every \( x \), an \( n \)-dimensional vector.

\[ x^T Ax = [x_1 \ldots x_n]^T [a_{ij}] \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j. \]

The result is a \( 1 \times 1 \) matrix. This is not easy to check in practice. Instead we appeal to the following:

**Theorem** if \( A \) is a positive definite \( n \times n \) matrix then

a) \( A \) is nonsingular,

b) \( a_{ii} > 0 \) for each \( i = 1, 2, n, \)

c) \( \max_{1 \leq k,j \leq n} |a_{kj}| \leq \max_{1 \leq i \leq n} |a_{ii}|, \)

d) \( (a_{ij})^2 < a_{ii} a_{jj} \) for each \( i \neq j \).

Further,
Theorem: A is symmetric positive definite if and only if Gaussian elimination can be performed without row exchange on $Ax = b$ with all pivots positive.

Corollaries:

1) A is positive definite if and only if $A = LDL^T$, $L$ is unit lower triangular $D$ is diagonal matrix $d_{ii} > 0$.

2) A is positive definite if and only if $A = LL^T$, $L$ lower triangular $l_{ii} \neq 0$.

12.1.7 Crout Factorization of a Tri-diagonal Linear System

Definition: An $n \times n$ matrix is called a “Banded Matrix” if $p$ and $q \in \mathbb{N}$ with $1 < p, q < n$ exist, with the property $a_{ij} = 0$ whenever $i + p \leq j$ or $j + q \leq i$. The bandwidth $bw$ is then defined as the number $bw \equiv p + q - 1$. For example the matrix with non-zero entries marked as $x$:

$$A = \begin{bmatrix} x & x & x & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 \\ 0 & 0 & x & x & x & x & 0 \\ 0 & 0 & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

has a bandwidth $bw = 6$.

The efficient storage of matrices is called “Packing.” It is almost always true that many of the zero entries in a matrix needn’t be stored. A matrix with a small bandwidth will thus require little storage, enabling one to consider bigger problems on any finite-memory machine. Matlab makes this very easy to accomplish via its `sparse` format command for vectors and matrices. Furthermore, there are important and often-occurring matrices with
small bandwidths have other nice properties. For example, strong diagonal dominance. Researchers spend much time optimizing codes that are storage efficient and also have nice mathematical properties which enable them to produce more efficient solvers.

Example: Two common small bandwidth matrices which come up in the solution of partial differential equations, for example, is the tri-diagonal and penta-diagonal matrices.

Tridiagonal Matrix (Banded Matrix) will have the structure:

\[
\begin{bmatrix}
  a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\
  a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\
  0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\
  \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\
  0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn}
\end{bmatrix}
\]

here \( p = q = 2 \), thus \( bw = 3 \).

If this matrix appears in a linear-algebraic problem \( Ax = b \), the equations are thus

\[
E_1 : a_{11}x_1 + a_{12}x_2 = b_1 = a_{1,n+1} \\
E_2 : a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 = a_{2,n+1} \\
\vdots \\
E_n : a_{n,n-1}x_{n-1} + a_{nn}x_n = b_n = a_{n,n+1}
\]

Note: the array \( b \) is stored in the \( n + 1 \) column of the extended matrix \( A \), of size \( n \times (n + 1) \).

\[ \square \]

The Thomas Algorithm

If this matrix is diagonally dominant, then it is very easy to solve the problem \( Ax = b \), for \( x \):

The algorithm is called the “Thomas Algorithm” and it is the result of the Crout factorization: formally, write \( A = LU \), where \( U \) is upper-triangular.
and $L$ is unit-lower triangular. Then the forward problem is to solve $Ly = b$, the the backward problem is to solve $Ux = y$. These are two simple solves, since the underlying matrices are triangular, with a diagonal and a super/sub diagonal.

Generalizations of the Thomas algorithm exist for the penta-diagonal case. The tri- or penta- systems with a non-zero entry in the upper right hand corner and lower left hand corner (these are Jacobi matrices) are trivially obtained by using the Thomas algorithm. These Jacobi matrices occur frequently in the solution of partial differential equations with periodic boundary conditions.

**Input:** $n$, entries of $A$. The vector $b$ is stored in the $n+1$ column of $A$.

**Output:** solution $x_1, \ldots, x_n$.

**Algorithm:**

**Step 1:** Set $\ell_{11} = a_{11}$

$u_{12} = a_{12}/\ell_{11}$

**Step 2:** for $i = 2 \ldots n - 1$

set $\ell_{i,i-1} = a_{i,i-1}$th row of $L$

$\ell_{ii} = a_{ii} - \ell_{i,i-1}u_{i-1,i}$

$u_{i,i+1} = a_{i,i+1}/\ell_{ii}$ (i + 1) column of $U$

**Step 3:**

$l_{n,n-1} = a_{n,n-1}$

$l_{n,n} = a_{nn} - l_{n,n-1}u_{n-1,n}$

%Solve $Lz = b$:

**Step 4:** $z_1 = a_{1,n+1}\ell_{11}$

**Step 5:** for $i = 2, \ldots, n$

$z_i = \frac{1}{\ell_{ii}}[a_{i,n+1} - \ell_{i,i-1}z_{i-1}]$

%Solve $Ux = z$

**Step 6:** $x_n = z_n$

**Step 7:** for $i = n-1, \ldots, 1$

$x_i = z_i - u_{i,i+1}x_{i+1}$

**Step 8:** output $(x_1, \ldots, x_n)$

STOP
12.2 Iterative Methods for Solving Algebraic Systems

Clearly we need alternatives to direct methods for solving linear systems, when the system is large and/or it is ill-conditioned. We will learn about two basic strategies: iterative techniques, and preconditioning. Both of these techniques produce equivalent (or approximate) linear-algebraic problems. Iterative techniques produce a fixed point problem in the n-dimensional space of solutions (where n is the size of the square matrix $A$). Like the scalar counterpart case, there may be several ways to propose a fixed point problem, given $Ax = b$. The key is to choose the fixed point problem that leads to fastest convergence. The preconditioning idea is to find another matrix $B$, say, so that $BX = Bb$ is better conditioned than the original problem; better yet if $B$ is a good approximation to $A^{-1}$. We can also combine iteration with preconditioning.

We return to our favorite problem, but let us generalize to the possibly nonlinear algebraic problem: we want to solve

$$A(v) - b = 0,$$

where $A$ is a nonlinear discrete operator, $b$ is a known vector, and the discrete solution is $v$. (The problem, as posed, is more general than a set of nonlinear algebraic equations, but let’s focus on algebraic equations in finite dimensions.)

Often the solution to (76) is difficult to obtain directly, but the residual error

$$r = A(w) - b$$

for an approximate solution $w$ is easy to evaluate. If there is a related system

$$P(w) - b = 0$$

that approximates (76) and is easier to solve, the defect correction algorithm may be appropriate: Given a guess $v_n$ near a root $v_{n+1}$ of (76) we can expand this equation using a Taylor series to get:

$$0 = A(v_{n+1}) - b$$

$$= A(v_{n+1}) - b + P(v_{n+1}) - P(v_{n+1})$$

$$(77) = A(v_n) - b + P(v_{n+1}) - P(v_n) - (J_P - J_A)(v_{n+1} - v_n) + O(\epsilon^2),$$
Table 4: Common Examples of Defect Correction Iteration.

<table>
<thead>
<tr>
<th>$P(v_{n+1})$</th>
<th>METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(v_n) + J_A(v_n)(v_{n+1} - v_n)$</td>
<td>Newton</td>
</tr>
<tr>
<td>Diagonal $J_A$</td>
<td>Jacobi</td>
</tr>
<tr>
<td>Lower triangular part of $J_A$</td>
<td>Gauss-Seidel</td>
</tr>
<tr>
<td>Lower triangular part of $J_A$ plus first upper off-diagonal</td>
<td>Line Gauss-Seidel</td>
</tr>
<tr>
<td>Coarse grid operator plus relax using one of the above</td>
<td>Multigrid</td>
</tr>
<tr>
<td>Symmetric part of $J_A$</td>
<td>Concus-Golub-O’Leary</td>
</tr>
<tr>
<td>If $\Lambda = (I + \Delta tL_x + \Delta tL_y)$, the $P = (I + \Delta t\tilde{L}_x + \Delta t\tilde{L}_y)$, where $\tilde{L}_x$ is the linearized lower order approximation of $L$.</td>
<td>ADI</td>
</tr>
<tr>
<td>$LU$, where $L$ lower $U$ upper triangular</td>
<td>incomplete LU</td>
</tr>
</tbody>
</table>

where $\epsilon = v_{n+1} - v_n$. The defect correction iteration is any $O(\epsilon)$ approximation to (77).

The simplest such iteration is

$$P(v_{n+1}) = P(v_n) = A(v_n) + b.$$ (78)

This iteration will converge if $v_n$ and $J_P$ the Jacobian of $P$ are near enough to $v_{n+1}$ and $J_A$, respectively. Table 4 lists some of the more common applications of defect corrections.

The iteration of (78) can often be sped up by using a one-step acceleration parameter $\omega$ to give

$$P(v_{n+1}) = P(v_n) - \omega_n[A(v_n) - b].$$

These accelerated methods include SOR, dynamic alternating direction implicit methods, and damped Newton. Often a 2-step acceleration method

$$P(v_{n+1}) = b + \omega_n[P(v_n) - \alpha_nA(v_n)] + (1 - \omega_n)P(v_{n-1})$$

can speed up the convergence even more. These methods include Chebyshev and Conjugate Gradient methods.

258
12.2.1 Newton’s Method for Nonlinear Systems

Solving nonlinear systems iteratively is an active area of research, and there are a number of good sources to consult\(^{10}\). Some of the most common methods are:

1. Steepest Descent
2. Conjugate Gradient (see 12.2.3)
3. Broyden and variable metric methods (see BFGS at netlib)
4. Fixed Point Methods
5. Method of Weighted Residuals
6. Tchebyshev acceleration

This list is not comprehensive, of course, but perhaps the most common method is “Newton Raphson’s Method” which we will do next. It is also a good method to understand since many of its pros and cons are the basis for the development of other methods. Let \( f(x) \in C^2 \) be an \( n \)-dimensional vector of functions that depend on \( x \in \mathbb{R}^n \). The problem we want to solve is the root-finding problem

\[
  f(x) = 0.
\]

That is,

\[
  f_i(x_1, \ldots, x_n) = 0 \quad 1 \leq i \leq n,
\]

\[
  x = (x_1, x_2, \ldots, x_n)^T,
\]

\[
  f = (f_1, f_2, \ldots, f_n)^T.
\]

Just as we did in the scalar case, we expand \( f \) about \( x \), locally (see 5.3) and retain to first order in \( ||H|| \):

\[
  0 = f(x + H) \approx f(x) + J(x)H + O(||H||^2).
\]

\(^{10}\)The book by Reinhardt, Varga, et al. and the book on Optimization by Nocedal and Wright are good starting places for the basics and a survey, respectively
Here, the Jacobian Matrix is

\[ J(x) \equiv f'(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \cdots & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}. \]

Hence, we can solve formally for the perturbation:

\[ H = -[J(x)]^{-1}f(x), \]

it is now straightforward to see that the situation is not different from the scalar case. We form an iterative problem: iteratively solve for \(x^{(k+1)}\), for \(k = 0, 1, 2, \ldots\), starting with some initial guess \(x^0\). This can be done by solving for the update \(H^k\), using

\[ J(x^k)H^k = -f(x^k), \]

which is a linear-algebraic problem, with an update

\[ x^{(k+1)} = x^k + H^{(k)}. \]

Convergence is now measured in terms of vector norms and the conditions for success or failure are more complex, but more geometrical. The same algorithmic issues discussed in the scalar case apply here (see 5.3). The new issue is: How to solve the linear algebraic problem. We know how to solve these types of problems with direct methods (see Section 12.1). But as we already know, direct methods are not always the most efficient technique, and in some instances, they are computationally prohibitive. So we study next iterative techniques for the solution of linear systems.

12.2.2 Iterative Techniques for Solving Linear Systems

We’ll consider here three closely associated elementary methods: Jacobi (12.2.2), Gauss-Seidel (12.2.2), and Successive over-relaxation (SOR) (12.2.2). We will then discuss briefly Conjugate Gradient (for the solution of linear systems)(see 12.2.3), and later, the Multigrid method\(^{11}\).

\(^{11}\)Also see Briggs, et al., *A Multigrid Tutorial* for a nice introduction to the subject.
We want to solve

\[ Ax = b, \]

here \( x \) and \( b \) are in \( \mathbb{R}^n \), and \( A \) is an \( n \times n \) matrix. The basic idea in Jacobi, Gauss-Seidel and SOR is to solve for a sequence \( \{x^{(k)}\}_{k=0}^{\infty} \) that converges to \( x \), starting with an initial guess \( x^0 \), of \( x \). This technique is especially well suited for large matrix problems that have no particular structure to them (e.g., they are not symmetric positive definite, or normal, etc). They are reasonably fast if the problem is well-conditioned. Moreover, they can be made very efficient if \( A \) has lots of zeros.

We convert (79) into

\[ x = Tx + c, \]

which we recast as an iterative scheme

\[ x^{k+1} = Tx^k + c, \quad k = 0, 1, \ldots, \]

with a starting guess \( x^{(0)} \). There’s no coincidence in (81) looking like an iterative fixed point problem. Since we know a lot about fixed point iteration we also will know how this iteration will perform and whether it has a unique fixed point. We just have to work in \( n \)-dimensional space. We also know from our experience with fixed point iteration that there are potentially several ways of recasting \( f = Ax - b = 0 \), the root-finding problem.

Example Suppose \( A \) is

\[
\begin{pmatrix}
-1 & 0.1 & 0.5 \\
0.2 & 10 & 2 \\
0.5 & 1 & -3
\end{pmatrix},
\]

and \( b = [-0.3, 22.2, -0.5]^T \) (here the superscript \( T \) is “transpose”). The solution of (79) is

\[ x = [1, 2, 1]^T. \]

To cast (79) as an iterative map (82), we could propose

\[
\begin{align*}
x_1 &= \frac{1}{10} x_2 + \frac{1}{2} x_3 + \frac{3}{10} \\
x_2 &= -\frac{1}{50} x_1 - \frac{1}{5} x_3 + \frac{222}{100} \\
x_3 &= \frac{1}{6} x_1 + \frac{1}{3} x_2 + \frac{1}{6}.
\end{align*}
\]

261
This is obtained by solving for \( x_i, i = 1, 2, 3 \) in each of the row equations.

Then \( T = \begin{bmatrix} 0 & 10 & 1 \frac{1}{2} \\ -\frac{1}{50} & 0 & -\frac{1}{5} \frac{1}{2} \end{bmatrix} \), and \( c = \begin{bmatrix} \frac{3}{100} \\ \frac{1}{60} \end{bmatrix} \).

Take \( x^{(0)} = [0, 0, 0]^T \) as the initial guess. Solving above generates \( x^{(1)} = [0.9000, 2.0000, 0.6667]^T \). Then use this as a next guess and find \( x^{(2)} \), and so on. You’ll find that you need about 10 iterations to get \( x^{10} = [1.0000, 2.0000, 1.0000]^T \).

So that
\[
\frac{||x^{10} - x^9||_\infty}{||x^{10}||_\infty} = 8.0 \cdot 10^{-4} < 10^{-3}. \text{ In fact } ||x^{10} - x||_\infty = 0.0002.
\]

This is the idea behind Jacobi iteration. So let’s formalize it:

**Jacobi Iteration**

\[
(82) \quad \text{Solve } x_i = \sum_{j=1}^n \left( \frac{-a_{ij}x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}} \quad i = 1, 2, \ldots, n.
\]

Next, generate a sequence: for \( k \geq 1 \ldots,
\[
x_i^{(k)} = \frac{1}{a_{ii}} \left\{ \sum_{j=1}^n \left[ 1 - a_{ij}x_j^{(k-1)} + b_j \right] \right\} \quad i = 1, 2, \ldots, n
\]

Compactly:
\[
A = D - L - U =
\]

\[
\begin{bmatrix}
a_{11} & 0 \\
a_{22} & 0 \\
\vdots & \ddots \\
0 & a_{nn}
\end{bmatrix} -
\begin{bmatrix}
0 & \cdots & 0 \\
-a_{21} & 0 \\
\vdots & \ddots \\
-\frac{a_{n1}}{a_{11}} & \cdots & -\frac{a_{n,n-1}}{a_{nn}} & 0
\end{bmatrix} -
\begin{bmatrix}
0 & -a_{12} & \cdots & -a_{1,n} \\
0 & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & -a_{n-1,n}
\end{bmatrix}.
\]

Thus, the problem \( Ax = b \) is recast as
\[
Dx = (L + U)x + b.
\]

262
Solving,

\[ x = D^{-1}(L + U)x + D^{-1}b. \]

The *Jacobi Iterative Procedure* is thus,

\[ x^{k+1} = D^{-1}(L + U)x^k + D^{-1}b, \quad k = 0, 1 \ldots, \text{with starting } x^{(0)}. \]

The Jacobi Algorithm

1. **input:** \( n, a_{ij}, b_i; \ X_i, \) TOL, NMAX (maximum number of iterations)
2. **output:** approximation of \( x \) or failure.
3. **Step 1** \( k = 1 \)
4. **Step 2** While \( (k \leq NMAX) \) do 3–6
5. **Step 3** for \( i = 1 \ldots n \)
6. \( x_i = - \sum_{j=1}^{n} a_{ij} X_j + b_i/a_{ii} \)
7. **Step 4** if \( ||x - X|| < \text{TOL} \) then output \((x_1 \ldots x_n) \) STOP
8. **Step 5** \( k = k + 1 \)
9. **Step 6** for \( i = 1 \ldots n \) set \( X_i = x_i \)
10. **Step 7** output ('Max iterations exceeded')

**END**

Note \( a_{ii} = 0 \) must be avoided by re-ordering \( A \). To speed up convergence \( a_{ii} \) should be as large as possible, hence, rebalancing via scaling is often also worthwhile.

One can use \( \frac{||x^{(k)} - x^{(k-1)}||}{||x^{(k)}||} < \text{TOL} \) in step 4 for stopping, also, might want to also examine \( ||b - Ax^{(k)}|| \). Usually the sup-norm is used.

The Gauss-Seidel Method
A variant of Jacobi:

Replace (82) by

\[ x^{(k)}_i = -\sum_{j=1}^{i-1} (a_{ij}x^{(k)}_j) - \sum_{j=i+1}^{n} (a_{ij}x^{(k-1)}_j) + b_i, \]

for each \( i = 1, 2, \ldots, n \).

So, the idea is this: since \( x^{(k)}_i \) are to be computed, and these use \( x^{(k-1)}_i \), why not use \( x^{(k)}_1 \cdots x^{(k)}_{i-1} \). These are likely a better approximation than \( x^{(k-1)}_1 \cdots x^{(k-1)}_{i-1} \). So we use these the entries of the \( k^{th} \) iterate whenever these become available.

Compactly, Gauss-Seidel is obtained by iterating

\[ (D - L)x^{(k)} = Ux^{(k-1)} + b. \]

For \( D - L \) to be non-singular it is necessary and sufficient that \( a_{ii} \neq 0 \).

The Gauss-Seidel Algorithm

Same input/output as Jacobi.

Step 1 \( k = 1 \)
Step 2 While \( k \leq NMAX \) do 3 - 6
Step 3 for \( i = 1 \ldots n \)
\[ x_i = \left[ -\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j + b_i \right] / a_{ii} \]
Step 4, 5, 6, 7, same as Jacobi.

Remark: There are systems for which Jacobi converges and Gauss-Seidel doesn’t, and vice versa. For specific linear systems \( Ax - b = 0 \), this can be shown by fixed point iteration theorems. We’ll discuss this issue more generally now:

General Theory of Iterative Methods

Let’s study methods of the form

\[ x^{(k)} = Tx^{k-1} + c, \quad k = 1, 2, \ldots \]

\( x^{(0)} \) the initial guess.

264
To study convergence we recall the following fact:

**Lemma** (Geometric Series, the infinite series case) If $\rho(T) < 1 \Rightarrow (I - T)^{-1}$ exists and

\[(I - T)^{-1} = I + T + T^2 + \cdots \tag{83}\]

**Proof:** if $\lambda$ is an eigenvalue of $T$ then $1 - \lambda$ is an eigenvalue of $I - T$. Since $|\lambda| \leq \rho(T) < 1$, therefore, no eigenvalue of $(I - T)$ is zero and $I - T$ is nonsingular.

Let $S_m = I + T + T^2 + \ldots + T^m$. Thus, $(I - T)S_m = I - T^{m+1}$. This is true for any matrix $T$.

Since $T$ (i.e. $\lim_{n \to \infty} T^n = 0$ is convergent then

\[
\lim_{m \to \infty} (I - T)S_m = \lim_{m \to \infty} (I - T^{m+1}) = I
\]

therefore,

\[
\lim_{m \to \infty} S_m = (I - T)^{-1}.
\]

To prove that $\lim_{n \to \infty} T^n = 0$ is not so simple. However, the following fact is illuminating: assume that $\rho(T) \geq 1$, then, from the eigenvalue problem

\[T^m x = \lambda^m x\]

(here, $x \neq 0$) it is clear that it is not possible that $T^m \to 0$ as $m \to \infty$, as that would imply that $T^m x \to 0$.

**Corollary** (More geometric series) Let $T$ be as above and $\|T\| < 1$, then $(I - T)^{-1}$ exists and has the geometric series expansion (83). Moreover,

\[\|(I - T)^{-1}\| \leq \frac{1}{1 - \|T\|}.\]

**Proof:** use the geometric series theorem.

**Remark:** the above results go over to operators in Banach spaces. It is a useful result and one to remember.

**Theorem** $x^{(0)} \in \mathbb{R}^n$, the $\{x^{(k)}\}_{k=0}^\infty$ defined by

\[x^{(k)} = Tx^{(k-1)} + c, \quad k \geq 1 \quad c \neq 0\]

265
converges to the unique solution of

\[ x = Tx + c \]

if and only if \( \rho(T) < 1 \).

**Proof:** Use induction to get \( x^{(k)} = T^k x^{(0)} + (T^{k-1} \cdots T + I) c. \)

If \( \rho(T) < 1 \) \( \Rightarrow \lim_{k \to \infty} T^k x^{(0)} = 0. \)

Use previous Lemma and show that \( x - x^{(k)} = 0 \) as \( k \to \infty. \)

**Corollary** If \( ||T|| < 1 \) in some induced norm \( \Rightarrow \{ x^{(k)} \} \) converges for any \( x^{(0)} \in \mathbb{R}^n \) to \( x \in \mathbb{R}^n \) and

(i) \( ||x - x^{(k)}|| \leq ||T||^k ||x^0 - x||, \)

(ii) \( ||x - x^{(k)}|| \leq \frac{||T||^k}{1 - ||T||} ||x^1 - x^{(0)}||. \)

**Theorem:** If \( A \) is strictly diagonal dominant then for any \( x^{(0)} \) Jacobi and Gauss-Seidel converge to the unique solution of \( Ax = b. \)

**Remark:** Which method to pick? Since \( ||x^{(k)} - x|| \approx \rho(T)^k ||x^{(0)} - x|| \) choose method that gives smallest \( \rho(T) < 1. \)

For arbitrary systems? No general result regarding which is faster. Some guidelines:

**Theorem:** If \( a_{ij} \leq 0 \) for each \( i \neq j \) and \( a_{ii} > 0, \quad i = 1, 2 \ldots \) then one and only one of the following statements holds:

(a) \( 0 \leq \rho(T_G) < \rho(T_J) < 1, \)

(b) \( 1 < \rho(T_J) < \rho(T_G), \)

(c) \( \rho(T_J) = \rho(T_G) = 0, \)

(d) \( \rho(T_J) = \rho(T_G) = 1, \)

266
where $T_J$ is the matrix resulting from Jacobi and $T_G$ the matrix resulting from Gauss-Seidel.

\[ T_J \text{ and } T_G \]

Let’s re-examine the Gauss-Seidel method. A different look at the problem leads to the introduction of a relaxation parameter, and the generation of a more general class of iterative solvers, all identical except for the value of the parameter.

Let $\tilde{x} \in \mathbb{R}^n$ be approximation to $x$ of solution to $Ax = b$. Let $r = b - Ax$.

In Gauss-Seidel, in terms of the entries, $x_i^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_i^{(k-1)}, x_i^{(k)}, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)})^T$.

Also, let $r_i^{(k)} = (r_{1i}^{(k)}, r_{2i}^{(k)}, \ldots, r_{ni}^{(k)})^T$.

The $m^{th}$ component of $r_i^{(k)}$ is

\[ r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj}x_j^{(k)} - \sum_{j=i+1}^{n} a_{mj}x_j^{(k-1)} - a_{mi}x_i^{(k-1)}, \]

for each $m = 1, 2, \ldots, n$. In particular,

\[ r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} - a_{ii}x_i^{(k-1)}, \]

So

\[ a_{ii}x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}. \]

But Gauss-Seidel:

\[ x_i^{(k)} = \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} \right] \]

So (79) can be written as

\[ a_{ii}x_i^{(k-1)} + r_{ii}^{(k)} = a_{ii}x_i^{(k)}. \]
Therefore Gauss-Seidel can be cast as the problem of choosing $x_i^{(k)}$ to satisfy

\[
x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}.
\]

(87)

What we want to do is to drive $r_{ii}$ to zero.

Another view on this: Consider $r_{i,i+1}^k$ associated with

$\mathbf{x}_i^{(k)} = (x_1^{(k)}, \ldots, x_i^{(k)}, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)})^T$. By (84)

\[
r_{i,i+1}^{(k)} = b_i - \sum_{j=1}^{i} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}
\]

\[
= b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} - a_{ii} x_i^{(k)}.
\]

Equation (86) implies that $r_{i,i+1}^{(k)} = 0$. In this sense Gauss-Seidel is also characterized by requiring that the $i^{th}$ component of $r_{i,i+1}^{(k)}$ be driven to zero.

Finding how to zero out one of the components of $\mathbf{r}$ is not hard. The challenge is to drive the whole residual to zero i.e., drive the norm of $r_{i,i+1}^{(k)}$ to zero.

A way to achieve this efficiently is to modify (86):

\[
x_i^{(k)} = x_i^{(k-1)} + \omega \frac{r_{ii}^{(k)}}{a_{ii}} \quad \text{“Relaxation Method”}
\]

\[
0 < \omega, \quad \text{is a parameter, a real number.}
\]

For $0 < \omega < 1$ we get “under-relaxation”. One can vary $\omega$ in the hopes of getting convergence for some choices where Gauss-Seidel fails to converge.

When $\omega > 1$ we get “over-relaxation”. This range is used to obtain faster convergence when Gauss-Seidel converges. So this can be codified in another method:

**Successive Over Relaxation (SOR)**

\[
x_i^{(k)} = (1 - \omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}} [b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)}] = (1 - \omega) a_{ii} x_i^{k-1} - \omega \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} + \omega b_i,
\]

268
Or more compactly,

\[ x_k = \frac{(D - \omega L)^{-1}[(1 - \omega)D + \omega U]}{T_\omega} x^{k-1} + \omega b. \]

How do we choose \( \omega \)? No clear way. However:

**Theorem:** if \( a_{ii} \neq 0 \) for \( i = 1, \ldots, n \), then \( \rho(T_\omega) \geq |\omega - 1| \), therefore \( \rho(T_\omega) < 1 \)
only if \( 0 < \omega < 2 \), where

\[ T_\omega = (D - \omega L)^{-1}[(1 - \omega)D + \omega U] \]

**Theorem:** if \( A \) is positive definite and \( 0 < \omega < 2 \) then SOR converges for any \( x^{(0)} \).

**Theorem:** If \( A \) positive definite tridiagonal then \( \rho(T_G) = [\rho(T_J)]^2 < 1 \). Then, the optimal choice \( \omega \) for SOR is

\[ \omega = \frac{2}{1 + \sqrt{1 - \rho(T_G)^2}} = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}}. \]

Then \( \rho(T_\omega) = \omega - 1 \)

**SOR and Gauss-Seidel Algorithm:**

**Input:** Same as before and the parameter \( \omega \)

**Output:** \( x \)

**Step 1** \( k = 1 \)

**Step 2** While \( (k \leq NMAX) \) do 3 - 6

**Step 3** for \( i = 1, \ldots, n \)

\[ x_i = (1 - \omega)X_i + \frac{\omega}{a_{ii}} \left( -\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}X_j + b_i \right) \]

**Step 4** if \( ||x - X|| < TOL \), output \( x \), END

**Step 5** \( k = k + 1 \)

**Step 6** for \( i = 1, \ldots, n \) \( X_i = x_i \)

**Step 7** FAIL MESSAGE, STOP.

END

269
Another look at convergence of iterative methods: We learn from here another way to prove whether an iterative scheme converges.

We write Jacobi, Gauss-Seidel, SOR as

\[ x^{(k+1)} = T x^{(k)} + c \]

The error at the \( k^{th} \) step is \( e^{(k+1)} = x - x^{(k)} \) where \( x \) is the solution of \( Ax = b \).

Hence \( e^{(k+1)} = T e^{(k)} \)

Using inductions \( e^{(k)} = T^k e^{(0)} \)

The sequence \( x^{(1)}, x^{(2)}, \ldots, x^{(k)}, \ldots \), will converge to \( x \) as \( n \) tends to infinity is \( \lim_{k \to \infty} e^{(k)} = 0 \).

Assume \( T \) is an \( m \times m \) matrix. Assume it has \( m \) linearly independent eigenvectors \( v_s, s = 1, 2, \ldots, m \), and write

\[ e^{(0)} = \sum_{s=1}^{m} c_s v_s \quad \text{where} \ c_s \ \text{are scalars. Hence,} \]

\[ e^{(1)} = T e^{(0)} = \sum_{s=1}^{m} c_s \lambda_s v_s \]

where \( \lambda_s \) are the eigenvalues of \( T \).

Similarly,

\[ e^{(k)} = \sum_{s=1}^{m} c_s \lambda_s^k v_s. \]

Therefore \( e^{(k)} \) tends to the null vector as \( k \to \infty \) for any \( e^{(k)} \) if and only if \( |\lambda_s| < 1 \) for all \( s \). Therefore the iteration converges if \( \rho(T) < 1 \). As a corollary: a sufficient condition for convergence is that \( ||T|| < 1 \) because \( \rho(T) \leq ||T|| \).

The Eigenvalues of the Jacobi and Gauss-Seidel and SOR Iterations Matrices

Assume that the \( n \) linear equations

\[ Ax = b \]

270
are such that the matrix $A = D - L - U$ is nonsingular and has nonzero diagonal elements $a_{ii}, \ i = 1, 2, \ldots, n$.

Jacobi Iteration:

The eigenvalues $\mu$ of the Jacobi iteration $T \equiv D^{-1}(L + U)$ are roots of

$$
\det[\mu I - D^{-1}(L + U)] = \det D^{-1}(\mu D - L - U)
$$

where $\det D^{-1} = 1/\det D = 1/a_{11}a_{22}\cdots a_{mm} \neq 0$.

Hence $\det(\mu D - L - U) = 0$.

SOR iteration and Gauss-Seidel Iteration

The $T \equiv (I - \omega D^{-1}L)^{-1}\{(1 - \omega)I + \omega D^{-1}U\}$

for which the eigenvalues $\lambda$ are roots of $\det(\lambda I - T) = 0$.

Now $\lambda I - T = \lambda(I - \omega D^{-1}L)^{-1}(I - \omega D^{-1}L) - (I - \omega D^{-1}L)^{-1}\{(1 - \omega)I + \omega D^{-1}U\}$

$= (I - \omega D^{-1}L)^{-1}\{(\lambda - \omega D^{-1}L) - (1 - \omega)I - \omega D^{-1}U\}$

$= (I - \omega D^{-1}L)^{-1}D^{-1}\{(\lambda + \omega - 1)D - \lambda \omega L - \omega U\}$

Therefore

$$
\det(\lambda I - T) = \det\{(\lambda + \omega^{-1})0 - \lambda \omega L - \omega U\} / \det(I - \omega D^{-1}L) \det D
$$

but $\det(I - \omega D^{-1}L) = \text{determinant of a unit lower triangular matrix} = 1$.

Also $\det D = a_{11}a_{22}\cdots a_{mm} \neq 0$.

Hence, the eigenvalues $\lambda$ are roots of

$$
\det\{(\lambda + \omega - 1)D = \lambda \omega L - \omega U\} = 0.
$$

This result holds for any set of linear equations satisfying $\det A \neq 0$ and $\det D \neq 0$. The eigenvalues $\lambda$ of the Gauss-Seidel are given by setting $\omega = 1$(i.e. they are the roots $\lambda$ of $\det(D - \lambda L - U) = 0$.

Iterative Refinement

One would think that if $\tilde{x}$ is approximate to $x$ of $Ax = b$ and $r = b - Ax$, then if $||r||$ is small then $||x - \tilde{x}||$ is small. However, this is not always the case.
Example Let’s consider the following linear system:

\[
\begin{bmatrix}
1 & 2 & | & 3 \\
1.0001 & 2 & | & 3.0001 \\
\end{bmatrix}
\]  

It has a unique solution \( \mathbf{x} = (1, 1)^T \). Now, take \( \tilde{\mathbf{x}} \equiv (3, 0)^T \). We compute \( ||\mathbf{r}||_\infty = 0.0002 \). Although small, \( \tilde{\mathbf{x}} \) is a poor approximation to \( \mathbf{x} \), i.e. \( ||\mathbf{x} - \tilde{\mathbf{x}}||_\infty = 2 \).

\[\Box\]

**Theorem:** Suppose \( A \) is nonsingular and \( \tilde{\mathbf{x}} \) approximation to \( \mathbf{x} \), the solution to \( A\mathbf{x} = \mathbf{b} \), then \( ||\mathbf{x} - \tilde{\mathbf{x}}|| \leq ||A^{-1}|| \||\mathbf{r}|| \) and \( \frac{||\tilde{\mathbf{x}} - \mathbf{x}||}{||\mathbf{x}||} \leq \kappa(A) \frac{||\mathbf{r}||}{||\mathbf{b}||} \), where \( \kappa(A) \) is the condition number for the matrix \( A \).

\[\Box\]

So, using the above example, \( ||A||_\infty = 3.001 \), and

\[
A^{-1} = \begin{bmatrix}
-10000 & 10000 \\
5000.5 & -5000 \\
\end{bmatrix}
\]

\( ||A^{-1}||_\infty = 20000 \) and \( \kappa(A) = 60002 \).

Note: Since \( \mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}} = A\mathbf{x} - A\tilde{\mathbf{x}} \), then \( ||\mathbf{b}|| \leq ||A|| ||\mathbf{x}|| \), then \( \mathbf{b}(\mathbf{x} - \tilde{\mathbf{x}}) = A\mathbf{x}A^{-1}\mathbf{r} \).

\[\Box\]

**Remark:** There are iterative refinement techniques for ill-conditioned systems. We don’t cover this here, but the basic idea is this: we solve \( A\mathbf{y} = \mathbf{r} \) iteratively, with \( \mathbf{r} = \mathbf{b} - A\mathbf{x} \).

\[\text{where } \mathbf{x} \approx \tilde{\mathbf{x}} + \tilde{\mathbf{y}} \text{ till we get a better } \mathbf{y}.\]

Then solve for \( \tilde{\mathbf{x}} \)

\[\Box\]

### 12.2.3 The Conjugate Gradient Method

There are a number of extremizing techniques for the solution of linear and nonlinear algebraic problems. Among the most general techniques are the
"Quasi-Newton" methods (you will find codes for these at netlib). These are fairly efficient and when contractive-mapping restrictions are met, they are fairly robust. A very popular quasi-Newton technique is known as the Broyden Method, and the Variable Metric Broyden (such as the BFGS).

Here we consider instead a “Steepest Descent” method, aimed at solving linear algebraic problems of a specific type. In steepest descent methods we construct a functional which when extremized will deliver the solution of the problem in question. The functional will have convexity properties, so that the vector which extremizes the functional is the solution of the algebraic problem in question. The convexity properties of the functional depend on the underlying linear algebraic equation.

In steepest descent we are in search of a vector for which the gradient of the functional is zero. The search is done in an iterative fashion, guided by the geometry of the gradient of the functional.

When the functional is globally convex, the gradient information provides us the direction to the minima. The \textit{Conjugate Gradient} method is applicable to such problems. As we will see, the search technique is amazingly efficient, but it is applicable to the solution of problems $A\mathbf{x} = \mathbf{b}$, where $A$ is symmetric positive definite (the negative definite case can trivially be recast as a positive definite case).

We want to solve

\begin{equation}
A\mathbf{x} = \mathbf{b} \quad \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}.
\end{equation}

Suppose $A$ is symmetric positive definite

\begin{equation*}
\left\{
\begin{array}{l}
A^T = A \\
x^T A x > 0, \quad x \neq 0.
\end{array}
\right.
\end{equation*}

\textbf{Lemma:} If $A$ is symmetric positive definite then solving $A\mathbf{x} = \mathbf{b}$ is equivalent to problem of minimizing the quadratic form

\[ f(x) = \frac{1}{2} x^T A x - x^T \mathbf{b} \]

where $f \in \mathbb{R}^n$. The minimum is reached when $\mathbf{x} = A^{-1} \mathbf{b}$.

\[ \square \]

$f$ is differentiable, and in fact

\[ -\nabla f(x) = -A \mathbf{x} + \mathbf{b}. \]
So looking for an extrema of $f$ means solving for $-\nabla f = 0$. Moreover, $\nabla^2 f = A$, and since $\|A\| > 0$ then the extrema is a minima.

Hence, we recast the problem of finding a solution of the (88) as the optimization problem: find $\hat{x}$ such that

$$f(\hat{x}) = \min_{x \in \mathbb{R}^n} f(x),$$

which is equivalent to finding $\hat{x}$ the solution of (88). Next we observe that

$$f(x) = \frac{1}{2}(x - \hat{x})^T A(x - \hat{x}) - \frac{1}{2} \hat{x}^T A \hat{x}$$

and minimization is the same as that of

(89) $$\hat{f} \equiv \frac{1}{2}(x - \hat{x})^T A(x - \hat{x})$$

since $f$ and $\hat{f}$ differ by a constant independent of $x$. But minimizing $\hat{f}$ is nonsensical since it involves $\hat{x}$ the unknown in the computation.

However, since

$$A(x - \hat{x}) = (A x - b) - (A \hat{x} - b) = r$$

where $r(x) \equiv A x - b$. Note that $r(\hat{x}) = 0$. Thus

$$f = \frac{1}{2} r^T A r$$

and it is clear then that

$$-\nabla f = b - A x = r.$$

Next, we recast the minimization as an iterative map: use $k$ as the iteration counter and let

$$x^{(k+1)} = x^{(k)} + \alpha_{k+1} p_{k+1}$$

which is called a line search, along the direction $p_{k+1}$. Here $\alpha$ is the search parameter and is used to minimize the functional $f(x^{(k)} + \alpha_{k+1} p_{k+1})$ along the direction $p_{k+1}$. This would be somewhat daunting in many space directions,
however, we can find $\alpha$ analytically: the minimum $\alpha_{k+1}$ occurs when the new residual is orthogonal to the search direction:

$$0 = \frac{d}{d\alpha_{k+1}} f(x^{(k+1)}) = \nabla f(x^{(k+1)})^T \frac{d}{d\alpha_{k+1}} x^{(k+1)}$$

$$= (Ax_{k+1} - b)^T (\frac{d}{d\alpha_{k+1}} (x^{(k)} + \alpha_{k+1} p_{k+1})) = -r_{k+1}^T P_{k+1}.$$

The new residual can be expressed in terms of the old residual:

$$r_{k+1} = b - Ax_{k+1}$$
$$= b - A(x^{(k)} + \alpha_{k+1} p_{k+1})$$
$$= b - Ax_{k} - \alpha_{k+1} A p_{k+1}$$
$$= r_k - \alpha_{k+1} A p_{k+1}.$$  

Thus, solving,

$$\alpha_{k+1} = \frac{r_k^T r}{P_{k+1}^T A P_{k+1}}.$$  

So we know how to get $\alpha$ and we can find $p$ by the requirement that it has to be orthogonal to $r$. But how do we find $p_{k+1}$ at each iterate $k$?

To proceed, we need to make the following definition:

A-Conjugate vectors:

A set of non-zero vectors $p_1, p_2, \ldots, p_n \in \mathbb{R}^n$ is said to be $A$-conjugate if

$$\delta_{ij} = p_i^T A p_j, \quad 1 \leq i, j \leq n. \quad (90)$$

The $p_j$ are called “conjugate directions.” Condition (90) is equivalent to requiring that $p_1, p_2, \ldots, p_n$ be an orthogonal basis for $\mathbb{R}^n$ with respect to the inner product $(\cdot, \cdot)_A$. Hence they are an $A$-conjugate complete set have linearly independent vectors. Also, (89) is no more than the norm $\frac{1}{2} \Vert \cdot \Vert_A^2$, i.e.

$$\tilde{f}(x) = \frac{1}{2} \Vert x - \tilde{x} \Vert_A^2.$$

Take

$$x^{(k)} = x^{(k-1)} + \alpha_k p_k, \quad (91)$$

275
\( k = 0, 1, \ldots \). Suppose we start (91) with \( x^{(0)} = 0 \). We can always do this: to see this, let \( Az = b - Ax^{(0)} \) with solution \( z^* \), then \( x^* = x_0 + z^* \) hence an initial guess \( z^{(0)} = 0 \) corresponds to \( x^{(0)} = x_0 \) in the original problem.

Express

\[
\tilde{x}^{(k)} = \sum_{j=1}^{n} \alpha_j p_j,
\]

there solution of \( Ax = b \). Multiply both sides of this equation by \( A \) and then by \( p_{k+1} \), then using orthogonality

\[
\alpha_{k+1} = \frac{p_{k+1}^T A \tilde{x}}{p_{k+1}^T A p_{k+1}} = \frac{p_{k+1}^T A b}{p_{k+1}^T A p_{k+1}}, \quad k = 0, 1, 2, \ldots n - 1.
\]

Note that when \( k = n \), \( x_n = \tilde{x} \) and \( r_n = 0 \). Hence, minimization requires at most \( n \) iterations. The rate, as we shall state later, depends on the eigenvalues of \( A \).

The conjugate gradient method gives a way to generate \( \{p_k\} \) and \( \{x_k\} \): Note that \( r_0 = b \), if \( x^0 = 0 \), and that \( r_k \) is orthogonal to \( p_i \), \( i = 1, 2, \ldots, n \), i.e.

\[
r_k \begin{pmatrix} T \\
p_i \end{pmatrix} = \delta_{k,i}.
\]

Starting with \( p_1 = b \) the conjugate gradient is then:

\[
\begin{align*}
\alpha_{k+1} &= \frac{r_k^T r_k}{p_{k+1}^T A p_{k+1}} \quad k \geq 0 \\
x^{(k+1)} &= x^{(k)} + \alpha_{k+1} p_{k+1} \quad k \geq 0 \\
\beta_{n+1} &= \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} \quad k > 0 \\
p_{k+1} &= r_k + \beta_{k+1} p_k \quad k > 0 \\
r_{k+1} &= b - A x^{(k+1)} \quad k \geq 0
\end{align*}
\]

The iterates converge to solution of \( Ax = b \) in at most \( n \) iterations!

Example Solve \( Ax = b \) with

\[
A = \begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
0 \\
1
\end{bmatrix}
\]

276
by the conjugate gradient method. Let $x^{(0)} = 0$, then $r_0 = p_1 = b = [1 0 1]^T$. Form $r_0^T r_0 = 2$ so

$$p_1^T A p_1 = [1, 0, 1] [a_{ij}] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 4$$

Therefore $\alpha_1 = r_0^T r_0 / p_1^T A p_1 = 0.5$ and

$$x^{(1)} = x^{(0)} + \alpha_1 p_1 = \begin{bmatrix} 0.5 \\ 0.0 \\ 0.5 \end{bmatrix}$$

For the next iteration $r_1 = b - A x^{(1)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

$$\beta_2 = r_1^T r_1 / r_0^T r_0 = \frac{1}{2} = 0.5 \therefore$$

$$p_2 = r_1 + \beta_2 p_1 = [0.5 1.0 0.5]^T$$

$$p_2^T A p_2 = 1 \Rightarrow \alpha_2 = 1 \text{ and } x^{(2)} = x^{(1)} + \alpha_2 p_2 = [1 1 1]^T$$

the residual $r_2 = 0$ and $x^{(2)}$ is the solution. \( \Box \)

A conservative estimate of the rate of convergence of the conjugate gradient method is

$$\|x - \tilde{x}\|_A \leq 2 \left[ \frac{1}{1 + \sqrt{c}} \right] \|\tilde{x}\|_A.$$  

Here, $c = \lambda_1 / \lambda_n$, where $0 < \lambda_1 \leq \lambda_2 \ldots \leq \lambda_n$ are the eigenvalues of $A$. Also, note that $c$ is the reciprocal of the condition number of $A$, estimated using the $L_2$-norm.

Remark: For a symmetric positive definite but ill-conditioned $A$, the Conjugate Gradient Method is very slow. What can be done in this case? we precondition.

Preconditioning Take $A x = b$
and it turn into

$$By = d.$$ 

where

$$B = Q^T AQ, \quad y = Q^{-1}x \quad \text{and} \quad d = Q^T b.$$ 

To choose $Q$: pick nonsingular matrix in such a way so that $\kappa(B) < \kappa(A)$, i.e. the condition of $B$ is smaller than $A$’s.

Careful analysis is required to choose a $Q$.

In some cases, for the preconditioned conjugate gradient method, one could reduce convergence cost to $O(\sqrt{n})$.

Remark: In preconditioned Conjugate Gradient the recast $By = d$ is actually not computed or used. There is a modification that allows us to use the above discussed conjugate gradient method (see Gollub and van Loan’s book, for example.

12.2.4 A Comparison of Convergence of Jacobi 12.2.2, Gauss-Seidel 12.2.2, SOR 12.2.2, and Conjugate Gradient 12.2.3

In the context of solving a specific example, for example in finding approximate solutions to $v(x, y)$, where

$$\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2} = -\Delta v = f(x, y)$$

over a square domain in the 2D-plane $x, y$, with $v$ specified everywhere on the boundary. (We will encounter this Elliptic Problem in 475B).

If we use a center-difference approximation to the derivatives, and we use a uniform grid for the domain, we will arrive at the associated linear algebraic problem:

$$Az = b$$

where $z$ is the vector representing $v_{x_i,y_j}$, with $i, j = 1, 2, ... n$, say, where $(x_i, y_j)$ represents the 2D lattice that results from the finite-differencing using $n$ grid points in $x$ and $y$. 278
So to solve the “Poisson Problem” amounts to solving the linear algebraic problem stated above.

If we denote the spectral radius of the Jacobi case as \( \rho(A_J) \) (see 11.2), we will find that

\[
\rho(A_J) \approx 1 - \frac{a}{n} \quad \text{as } n \rightarrow \infty \quad \text{number of unknowns} \quad \text{grows} \quad \rho \rightarrow 1.
\]

Since the error is multiplied by \( \rho \Rightarrow \) convergence slows down.

\( k \) iterations \( \approx O(n) \) to reduce error by \( e^{-1} \).

Let \( \rho(A_{GS}) \) be the spectral radius for the Since \( \rho(A_{GS}) = \rho^2(A_J) \Rightarrow 1 \) Gauss-Seidel can decrease as much as Jacobi. Now, let \( \rho(A_{SOR}) \) be the spectral radius for the SOR solution.

Then \( \rho(A_{SOR}) \approx 1 - \frac{a'}{\sqrt{n}} \) when optimal.

so \( SOR \) is approximately \( \sqrt{n} \) faster than either Jacobi or Gauss Seidel Methods. Now, The condition number for the Conjugate Gradient is \( \kappa = O(n) \). Hence, after \( k \) steps in the Conjugate Gradient Method the residual is multiplied by about \( [1 - O(\sqrt{n})]^k \), same as optimal SOR, thus the CG takes about \( O(\sqrt{n}) \).

\[\Box\]

13 NUMERICAL EIGENVALUE PROBLEMS

The calculation of eigenvalues of matrices is a particularly tricky affair, especially for very large matrices. It is a very active area of current research. Here we’ll introduce just a couple of classic methods: The power method, inverse iteration (it is based on the power method), Rayleigh-Ritz, and QR.

The power method is cheap and easy to learn, but very limited. But it teaches us something about functional iteration. Rayleigh-Ritz is essential in the sense that it is the basis of a lot of classical analysis in dynamics and quantum mechanics (e.g., finding the state of a system which is at the lowest energy state).
QR is really the only serious method presented in these notes. The two best presentations on this subject, for the beginning student, are those by Trefethen and Bau, and Strang’s presentation. They are primarily geometrically based, but this is what makes them so powerful. Another classic method that’s enjoying a revival is Arnoldi iteration, which is used in the estimation of eigenvalues of very large matrices. Need to add notes on this.

13.1 The Power Method

Used to estimate the dominant eigenvalue and its corresponding eigenvector.

There are two situations: the case when $A$ ($n \times n$ matrix) has $n$ linearly independent eigenvectors, and there’s a single dominant eigenvalue.

This one’s easy.

The second variant is not discussed here and considers the cases not covered by above assumptions. This one’s complicated. See reference to Demmel et al.

We’ll consider the first case.

First, we order the eigenvalues

(i) $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$.

(ii) There’s an associated set of basis eigenfunctions $\{u^{(1)}, u^{(2)}, \ldots, u^{(n)}\}$ for $\mathbb{C}^n$, such that

$$Au^{(i)} = \lambda_i u^{(i)} \quad (1 \leq i \leq n).$$

Let $x^{(0)}$ be any element of $\mathbb{C}^n$ such that when $x^{(0)}$ is expressed as a linear combination of the basis $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$, with the coefficient of $u^{(1)}$ not 0. Thus,

$$x^{(0)} = a_1 u^{(1)} + a_2 u^{(2)} + \cdots + a_n u^{(n)}, \quad a_1 \neq 0.$$

We form

$$x^{(1)} = Ax^{(0)},$$
$$x^{(2)} = Ax^{(1)},$$

280
\[ x^{(k)} = A x^{(k-1)}, \]

so that
\[ x^{(k)} = A^k x^{(0)}. \]

In what follows we’ll absorb all the coefficients \( a_j \) in the vectors \( u^{(j)} \) that they multiply. Hence, write (93) as
\[ x^{(0)} = u^{(1)} + u^{(2)} + \ldots + u^{(n)}, \]

by (95) we can write (94) as
\[ x^{(k)} = A^{(k)} u^{(1)} + A^{(k)} u^{(2)} + \ldots A^k u^{(n)}, \]

using (92)
\[ x^{(k)} = \lambda_1^k u^{(1)} + \lambda_2^k u^{(2)} + \ldots \lambda_n^k u^{(n)}. \]

Rewrite this as
\[ x^{(k)} = \lambda_1^k \left[ u^{(1)} + \left( \frac{\lambda_2^k}{\lambda_1} \right) u^{(2)} + \ldots \left( \frac{\lambda_n^k}{\lambda_1} \right)^k u^{(n)} \right] \]

Since \( |\lambda_1| > |\lambda_j| \) for \( 2 \leq j \leq n \) we see that the coefficients \( (\lambda_j/\lambda_1)^k \) tend to 0 and the quantity in brackets converges to \( u^{(1)} \) as \( k \to \infty \).

Let’s write (96) as
\[ x^{(k)} = \lambda_1^k [u^{(1)} + \varepsilon^{(k)}] \]

where \( \varepsilon^{(k)} \to 0 \) as \( k \to \infty \). In order to take ratios let \( \theta \) be any linear functional on \( \mathbb{C}^n \) for which \( \theta(u^{(1)}) \neq 0 \). Note: \( \theta(\alpha x + \beta y) = \alpha \theta(x) + \beta \theta(y) \), since linear, where \( \alpha, \beta \) are scalars and \( x \) and \( y \) vectors. Then
\[ \theta(x^{(k)}) = \lambda_1^k [\theta(u^{(1)}) + \theta(\varepsilon^{(k)})] \]

\[ r_k \equiv \frac{\theta(x^{(k+1)})}{\theta(x^{(k)})} = \lambda_1 \left[ \frac{\theta(u^{(1)}) + \theta(\varepsilon^{(k+1)})}{\theta(u^{(1)}) + \theta(\varepsilon^{(k)})} \right] \to \lambda_1 \]

\[ \square \]
Remark: Since direction of $x^{(k)}$ aligns more and more with $u^{(1)}$ as $k \to \infty$ the method also yields $u^{(1)}$ the eigenvector.

What to chose for $\theta$? It is arbitrary. For example, simply evaluate the $j^{th}$ component of any given vector. In practice, the algorithm should normalize the $x^{(k)}$ otherwise they may converge to 0 or become unbounded, i.e., become too small or too large (we don’t care about the magnitude).

**The Power Method Algorithm**

**Input:** n, A, x, m. % here x is an initial guess, % m is the number of iterations.

**Output:** 0 or $x \neq 0$.

```matlab
for k = 1:m
    y = Ax
    r = $\theta(y)/\theta(x)$
    x = $y/||y||$ % use sup-norm, for example.
end
```

Estimate of Relative Error in the Computation: One can show that, since 
\[
\lim_{k \to \infty} r_k = \lambda_1,
\]
\[
\frac{r_k - \lambda_1}{\lambda_1} = \left(\frac{\lambda_2}{\lambda_1}\right)^k c_k
\]
where the numbers $c_k$ form a bounded sequence.

Can also show:
\[
r_{k+1} - \lambda_1 = (c + \delta k)(r_k - \lambda_1)
\]
\[
|c| < 1 \text{ and } \lim_{k \to \infty} \delta_k = 0
\]
which implies that $r_k$ converges linearly.
Using this knowledge one can accelerate the procedure:

**Theorem (Aitken Acceleration)**

Let \( \{ r_n \} \) be a sequence of numbers that converges to a limit \( r \), then the sequence
\[
s_n = \frac{r_n r_{n+2} - r_{n+1}^2}{r_{n+2} - 2r_{n+1} + r_n}, \quad n \geq 0,
\]
converges to \( r \) faster if \( r_{n+1} - r = (c + \delta_n)(r_n - r) \) with \( |c| < 1 \), and \( \lim_{n \to \infty} \delta_n = 0 \).

Indeed, \( (s_n - r)/(r_n - r) \to 0 \) as \( n \to \infty \).

\[\square\]

### 13.2 Inverse Power Method

We know that if \( \lambda \) is an eigenvalue of \( A \) and if \( A \) is nonsingular then \( \lambda^{-1} \) is an eigenvalue of \( A^{-1} \).

This suggests a way to estimate the smallest eigenvalue of \( A \) using the power method: arrange eigenvalues as
\[
|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{n-1}| > |\lambda_n| > 0.
\]

This can be done since \( A \) is non-singular, 0 is not an eigenvalue. The eigenvalues of \( A^{-1} \) arranged:
\[
|\lambda_n^{-1}| > |\lambda_{n-1}^{-1}| \geq \ldots \geq |\lambda_1^{-1}| > 0.
\]

Then apply the power method to \( A^{-1} \). But we don’t compute \( A^{-1} \). Instead, we solve
\[
Ax^{(k+1)} = x^{(k)},
\]
for \( x^{(k+1)} \) by some efficient linear algebra solver. One could consider an \( LU \) factorization, since it only has to be done only once.

These two facts suggest a way to find the eigenvalue farthest to a given value \( \mu \). The *Shifted Matrix Power Method*. Here \( \mu \) is complex, generally.
The trick is to construct a matrix

\[ \hat{A} = (A - \mu I) \]

and then use the regular power method on \( \hat{A} \), i.e.,

\[ x^{(k+1)} = \hat{A}x^{(k)}. \]

Finally, we could consider the eigenvalue closest to \( \mu \). In this case we apply the inverse power method on \( \hat{A} \), i.e.

\[ \hat{A}x^{(k+1)} = x^{(k)} \]

The power method can be also rigged up to find a series of eigenvalues: the idea is to remove the eigenvalues once these are found. This is called deflation and must be used with caution since round-off becomes an issue that might swamp the numerical sensitivity of the procedure.

13.3 The Rayleigh-Ritz Method:

If \( A \) is symmetric one can use an old, but efficient method to find the largest/smallest eigenvalue and associated eigenvector. It is quadratically fast, as compared with the power method. For \( A \) symmetric the eigenvalues and eigenvectors of \( A \) are real. In classical and quantum mechanics, \( A \) can sometimes result from the careful discretization of a self-adjoint, or more generally, a Hermitian operator.

Let \( A \) be an \( n \times n \) Hermitian matrix. Let \( x \) be an \( n \)-dimensional complex vector. We indicate Hermitian with the symbol \( \dagger \). Operationally, if a matrix satisfies \( A^\dagger = A \), it is Hermitian and its complex conjugate transpose equal to itself. If \( A \) is a real matrix, this simplifies to \( A^T = A \). That is, self-adjoint.

We will indicate by an overbar the operation of taking the complex conjugate and transpose. If \( A \) and \( x \) are real, this operation would simply involve the transpose of the matrix or the vector.
Since \( A^\dagger = A \) the eigenvalues are real and can be organized as
\[
\lambda_{\text{min}} \equiv \lambda_1 \leq \lambda_2 \leq \lambda_3 \cdots \leq \lambda_n \equiv \lambda_{\text{max}}.
\]
We will see that the smallest and largest eigenvalues may be thought of as the solution to a constrained minimum and maximum problem, respectively.

**Theorem (Rayleigh-Ritz):** Let \( A \) as above and the eigenvalues ordered as above. Then
\[
\lambda_1 \mathbf{x}\mathbf{x} \leq \mathbf{x}A\mathbf{x} \leq \lambda_n \mathbf{x}\mathbf{x}, \quad \text{for all } \mathbf{x} \in \mathbb{C}^n.
\]
Furthermore,
\[
\lambda_{\text{max}} = \max_{\mathbf{x} \neq 0} \frac{\mathbf{x}A\mathbf{x}}{\mathbf{x}\mathbf{x}} = \max_{|\mathbf{x}|^2 = 1} \mathbf{x}A\mathbf{x},
\]
and
\[
\lambda_{\text{min}} = \min_{\mathbf{x} \neq 0} \frac{\mathbf{x}A\mathbf{x}}{\mathbf{x}\mathbf{x}} = \min_{|\mathbf{x}|^2 = 1} \mathbf{x}A\mathbf{x}.
\]

**Proof:** Since \( A = A^\dagger \) then there exists a unitary matrix \( U \) such that \( A = U\Lambda U^\dagger \), with \( \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \). For any \( \mathbf{x} \in \mathbb{C}^n \) we have
\[
\mathbf{x}A\mathbf{x} = \mathbf{x}U\Lambda U^\dagger \mathbf{x} = (U^\dagger \mathbf{x})^\dagger \Lambda (U^\dagger \mathbf{x}) = \sum_{i=1}^n \lambda_i |(U^\dagger \mathbf{x})_i|^2.
\]
Since \( |(U^\dagger \mathbf{x})_i|^2 \) is non-negative, then
\[
\lambda_{\text{min}} \sum_{i=1}^n \lambda_i |(U^\dagger \mathbf{x})_i|^2 \leq \mathbf{x}A\mathbf{x} = \sum_{i=1}^n \lambda_i |(U^\dagger \mathbf{x})_i|^2 \leq \lambda_{\text{max}} \sum_{i=1}^n \lambda_i |(U^\dagger \mathbf{x})_i|^2.
\]
Because \( U \) is unitary
\[
\sum_{i=1}^n |(U^\dagger \mathbf{x})_i|^2 = \sum_{i=1}^n |\mathbf{x}_i|^2 = \mathbf{x}\mathbf{x}.
\]
Hence,
\[
\lambda_1 \mathbf{x}\mathbf{x} = \lambda_{\text{min}} \mathbf{x}\mathbf{x} \leq \mathbf{x}A\mathbf{x} \leq \lambda_{\text{max}} \mathbf{x}\mathbf{x} = \lambda_n \mathbf{x}\mathbf{x}.
\]
These are sharp inequalities. If \( \mathbf{x} \) is an eigenvector of \( A \) associated with \( \lambda_1 \), then
\[
\mathbf{x}A\mathbf{x} = \lambda_1 \mathbf{x}\mathbf{x}.
\]
Same sort of argument holds for $\lambda_n$.

Furthermore, if $x \neq 0$ then
\[
\frac{\langle Ax, x \rangle}{\langle x, x \rangle} \leq \lambda_n.
\]
so
\[
(97) \quad \max_{x \neq 0} \frac{\langle Ax, x \rangle}{\langle x, x \rangle} = \lambda_n.
\]

Finally, since $x \neq 0$, then
\[
\frac{\langle Ax, x \rangle}{\langle x, x \rangle} = \frac{\langle x, A x \rangle}{\langle x, x \rangle}
\]
and
\[
\frac{\langle x, x \rangle}{\sqrt{\langle x, x \rangle}} = 1.
\]
Hence, (97) is equivalent to
\[
\max_{\|x\|^2 = 1} \langle Ax, x \rangle = \lambda_n.
\]

Same sort of arguments hold for $\lambda_1$, in the context of the minimum. \hspace{1cm} \Box

**Algorithm**

Now we will revert to the case of $A$ an $n \times n$ symmetric real matrix for the presentation of the algorithm.

Let $x$ be an $n-$dimensional real vector. Choose some initial guess $x^{(0)}$, and compute
\[
x^{(m+1)} = Ax^{(m)}
\]
then
\[
\lambda_1^{(m+1)} = \frac{\langle x^{m+1}, x^{(m)} \rangle}{\langle x^{(m)}, x^{(m)} \rangle} \quad m \geq 0
\]
where $(w,z) = \sum_{i=1}^{n} w_i z_i \quad w, z \in \mathbb{R}^n$ is the inner product.

In fact, by writing $x^{(m)} = \sum_{j=1}^{n} \alpha_j^{(m)} u_j$ then
\[
\lambda_1 = \frac{\sum_{j=1}^{n} |\alpha_j^{(m+1)}|^2 \lambda_j^{2m+1}}{\sum_{j=1}^{n} |\alpha_j^{(m)}|^2 \lambda_j^{2m}}
\]
hence, it is easy to see that
\[
\lambda_1^{(m+1)} = \lambda_1 \left[ 1 + \mathcal{O} \left( \left[ \frac{\lambda_2}{\lambda_1} \right]^{2m} \right) \right],
\]
which is quadratic convergence, an improvement over the previous method.

\[\square\]

### 13.4 The QR Method

For reasonably-sized matrices, this is the most efficient and widely used method. There’s very good code on NETLIB EISPACK to do this. For large matrices, people are loading into Arnoldi methods (see Lehoucq). This method appeared in 1961 (Francis). Here’s an elementary introduction.

#### 13.4.1 The QR Factorization

Given \( A \), there’s a factorization of the form
\[
A = QR,
\]
where \( R \) is upper triangular and \( Q \) is orthogonal. If \( A \) is real both \( Q \) and \( R \) can be found to be real.

The main idea in QR is that we’ll construct an iterative technique to find the eigenvalues using similarity transformations.

Orthogonal matrices accomplish the transformations in such a way that they will not worsen the condition number or stability of the eigenvalue of a non-symmetric to matrix. A special class of orthogonal matrices is known as Householder Matrices, and we’ll concentrate on these.

**Note:** The term *orthogonal matrix* should be restricted to real matrices. However, usage of the term has extended to complex matrices. But in the complex case the matrices should be understood as being *unitary*.

Let \( w \in \mathbb{C}^n \) with \( \|w\|_2 = \sqrt{w^*w} = 1 \). Define
\[
U := I - 2ww^*
\]

(98)

287
this is the general form of a “Householder Matrix.”

Example: Construct a Householder matrix of dimension \( n = 3 \):

\[
\begin{bmatrix}
1 - 2|w_1|^2 & -2w_1\bar{w}_2 & -2w_1\bar{w}_3 \\
-2\bar{w}_1w_2 & 1 - 2|w_2|^2 & -2w_2\bar{w}_3 \\
-2\bar{w}_1w_3 & -2\bar{w}_2w_3 & 1 - |w_3|^2
\end{bmatrix}.
\]

An important property of \( U \) is that it is Hermitian. To see this:

\[
U^\dagger = (I - 2ww^*)^\dagger = I^+ - 2(w^*)^\dagger = I - (2w^*)^* w^* = I - 2ww^* = U.
\]

\( U \) is also orthogonal. To see this:

\[
U^\dagger U = U^2 = (I - 2ww^*)^2 = I - 4ww^* + 4(ww^*)(ww^*) = I,
\]

because \( w^*w = 1 \) and the associative law applies. Then

\[
(ww^*)(ww^*) = w(w^*w)^*w = ww^*.
\]

The QR Factorization of a Matrix

\[ A = QR. \]

Take \( A \in \mathbb{C}^{n \times n} \). Let \( P_r = I - 2w^{(r)}w^{(r)}\dagger \quad r = 1, \ldots, n-1 \), where \( \dagger \) indicates adjoint.

It turns out that we want to enter zeros in the definition of the \( w \)'s:

(99) \( w^{(r)} = [0, 0, \ldots, 0, w_r, \ldots, w_n]^T \equiv [O_{r-1}, \hat{w}^T]^T \) with \( \hat{w} \in \mathbb{C}^{n-r+1} \)

Writing \( A \) in terms of its columns \( A_{*1}, \ldots, A_{*n} \) we have

\[
P_1A = [P_1A_{*1}, \ldots, P_1A_{*n}].
\]

288
Pick \( P_1 \) and \( w^{(1)} \) using the following construction: we want to use the Householder matrix to transform a nonzero vector into a new vector containing mainly zeros. Let \( b \neq 0 \) be given, \( b \in \mathbb{C}^n \) and suppose we want to produce \( U \) of the form such that \( Ub \) contains zeros in positions \( r + 1 \) through \( n \), for some \( r \geq 1 \). Choose \( w \) as in (99). Then the first \( r - 1 \) elements of \( b \) and \( Ub \) are the same.

Let’s write \( m \equiv n - r + 1 \)

(100) so we take \( w = \begin{bmatrix} O_{r-1} \\ v \end{bmatrix} \) and

(101) \[ A_{*1} = \begin{bmatrix} c \\ d \end{bmatrix}, \]

with \( c \in \mathbb{R}^{r-1} \) and both \( v, d \in \mathbb{R}^m \). Then our restriction on the form of \( Ub \) requires the \( r - 1 \) components of \( Ub \) and \( c \) be same, and

(102) \( (I - 2vv^T)d = [\alpha, 0, 0 \ldots 0]^T, \quad ||v||_2 = 1. \)

for some \( \alpha \). Since \( I - 2vv^T \) is orthogonal, the length of \( d \) is preserved, therefore

\[ |\alpha| = ||d||_2 = S, \]

and further,

(103) \[ \alpha = \pm S = \pm \sqrt{d_1^2 + d_2^2 \cdots + d_m^2}. \]

Let

\[ p = v^Td. \]

From (102) we can deduce that

(104) \[ d - 2pv = [\alpha, 0, \ldots, 0]^T. \]

Multiplication by \( v^T \) and use of \( ||v||_2 = 1 \) implies that

(105) \[ p = -\alpha v_1. \]

Substituting this into the first component of (104) gives

\[ d_1 + 2\alpha v_1^2 = \alpha, \]

therefore \( v_1 = \frac{1}{2} \left[ 1 - \frac{d_1}{\alpha} \right] \).
Choose sign of \( \alpha \) in (103) by
\[
\text{sign}(\alpha) = -\text{sign}(d_1).
\]
This choice maximizes \( v_1^2 \) and avoids loss of significance errors in calculation of \( v_1 \). Having \( v_1 \), we can obtain \( p \) from (105).

Returning to (104) and then using components \( 2 - m \):
\[
v_j = \frac{d_j}{2p} \quad j = 2, 3, \ldots, m.
\]
Once \( v \) is obtained we can then see that \( w \) and thus \( U \) are obtained. The operation count can be shown to be \( O((2m + 2)) \).

Now \( P_1A \) contains zero below the diagonal in its first column. Choose \( P_2 \) similarly, so that \( P_2P_1A \) contains zeros in its second column below diagonal. Note that \( P_1A \) and \( P_2P_1A \) have same elements in row 1 and column 1. So to get \( P_2 \) and \( \mathcal{w}^{(2)} \) replace \( A_1 \) in (101) by second column of \( P_1A \). Continue procedure till we obtain an upper triangular matrix
\[
R = P_{n-1}P_{n-2} \cdots P_1A.
\]
If at the \( r \)-step of procedure all elements below diagonal of column \( r \) are zero then choose \( P_r = I \) and go to next step. To complete the construction let
\[
Q^T = P_{n-1}P_{n-2} \cdots P_1,
\]
which is orthogonal. Then \( A = QR \).

### 13.4.2 Obtaining Eigenvalues from a QR Factorization

Now that we know how to obtain a \( QR \) factorization we return to getting the eigenvalues:

Assume \( A \) is real \( \Rightarrow Q, R \) can be found to be real. Let \( A_1 = A \) and define a sequence
\[
A_m = Q_mR_m, \\
A_{m+1} = R_mQ_m, \\
\text{for} \quad m = 1, 2, \ldots
\]
Since \( R_m = Q_m^T A_m \), then
\[
A_{m+1} = Q_m^T A_m Q_m.
\]

\( A_{m+1} \) is orthogonally similar to \( A_m, A_{m-1} \cdots A_1 \). The sequence \( \{A_m\} \) will converge either to a triangular matrix with the eigenvalues of \( A \) on its diagonal or to a near-triangular matrix from which the eigenvalues can easily be calculated. This is slow but a shifting technique accelerates things considerably (not discussed here).

\[\square\]

**Remark:** QR is very expensive even with shifting. So the first step is to “prepare” the matrix by making it simpler. If \( A \) is symmetric it can be reduced to a similar symmetric tridiagonal matrix. If not symmetric, it can be reduced to its equivalent *Hessenberg Matrix* by a unitary similarity transformations.

An upper Hessenberg Matrix matrix has the form:
\[
H = \begin{bmatrix} 
* & * & - & - & - & * \\
* & 0 & * & & & \\
0 & * & 0 & * & & \\
: & 0 & * & & & \\
: & 0 & * & & & \\
0 & 0 & 0 & & & 
\end{bmatrix}
\]
i.e. \( h_{ij} = 0 \) when \( 1 > j + 1 \).

Note that if \( A \) is tridiagonal it is also Hessenberg. However, if \( A \) is tridiagonal, an efficient way to calculate its eigenvalue is by using “Sturm sequences.” (See Golub and VanLoan)

**Convergence of QR:** Suppose \( A \) is real \( n \times n \) with eigenvalues \( \{\lambda_i\} \),
\[
|\lambda_1| > |\lambda_2| \cdots |\lambda_n| > 0.
\]

The iterates \( R_m \) of the QR method will converge to an upper triangular matrix \( D \) with \( \{\lambda_i\} \) as diagonal elements. If \( A \) is symmetric, the sequence \( \{A_m\} \) converges to a diagonal matrix. For the rate of convergence
\[
||D - A_m|| \leq c \max_i \frac{\lambda_{i+1}}{\lambda_i}.
\]
For matrices with eigenvalues not satisfying (106), the iterates $A_m$ may not converge to a triangular matrix. For $A$ symmetric the sequence $\{A_m\}$ converges to a block diagonal matrix:

$$A_m \to D = \begin{bmatrix} B_1 & 0 \\ B_2 & \ddots \\ 0 & B_r \end{bmatrix},$$

in which $B_i$ are simple elements or $2 \times 2$ matrices. Thus the eigenvalues of $A$ can be easily computed from those of $D$. 

\[ \Box \]

### 13.5 Inverse Iteration

Here’s another technique for finding approximate eigenvectors. Suppose $A$ has a Jordan canonical form which is diagonal:

$$P^{-1}AP = \text{diag} [\lambda_1, \lambda_2, \ldots, \lambda_n].$$

Let the columns of $P$ be denoted by $x_1, x_2, \ldots, x_n$. Then

$$Ax_i = \lambda_i x_i \quad i = 1, 2, \ldots, n.$$

Assume $||x_i||_\infty = 1$ (can always be done). Let $\lambda$ be an approximation to a simple eigenvalue $\lambda_k$ of $A$. Given an initial $z^{(0)}$, define $\{w^{(m)}\}$ and $\{z^{(m)}\}$ by

$$(A - \lambda I)w^{(m+1)} = z^{(m)},$$

$$z^{(m+1)} = \frac{w^{(m+1)}}{||w^{(m+1)}||_\infty}, \quad m \geq 0.$$

Note: Here we want $\lambda$ to be a “poor” guess of $\lambda_k$, since otherwise we get a severely ill-conditioned matrix $A - \lambda I$! So choose a “close” value.

More precisely: let

$$z^{(0)} = \sum_{i=1}^{n} \alpha_i x_i, \quad \alpha_k \neq 0.$$
According to the power method:

\[(108) \quad z^{(m)} = \sigma_m (A - \lambda I)^{-m} z^{(0)} \quad \frac{||(A - \lambda I)^{-m} z^{(0)}||}{\infty}.
\]

\[(109) \quad |\sigma_m| = 1 \text{ i.e. either } \pm 1.
\]

Substituting (107):

\[(110) \quad (A - \lambda I)^{-m} z^{(0)} = \sum_{i=1}^{n} \alpha_i \left[ \frac{1}{\lambda_i - \lambda} \right] x_i.
\]

Let \(\lambda_1 - \lambda = \varepsilon\), and assume \(|\lambda_1 - \lambda| \geq c > 0\) for \(i = 1, 2, \ldots, n \quad i \neq k\).

From (109) and (110)

\[(111) \quad z^{(m)} = \sigma_m \frac{x_k + \varepsilon^m \sum_{i\neq k} \alpha_i \left[ \frac{1}{\lambda_i - \lambda} \right] x_i}{||x_k + \varepsilon^m \sum_{i\neq k} \alpha_i \left[ \frac{1}{\lambda_i - \lambda} \right] x_1||_{\infty}}.
\]

If \(|\varepsilon| < c\) then

\[||\varepsilon^m \sum_{i\neq k} \alpha_i \left[ \frac{1}{\lambda_i - \lambda} \right] x_i||_{\infty} \leq \left[ \frac{\varepsilon}{c} \right]^m \sum_{i\neq k} |\alpha_i|,
\]

which tends to 0 as \(m \to \infty\). This fact, along with (111) shows that \(z^{(m)}\) converges to a multiple of \(x_k\). The convergence is linear, though. In its implementation, a sensible thing to do is to \(LU\)-factorize \(A - \lambda I\).

\[\square\]

## 14 DIFFERENCE EQUATIONS

Let \(V\) denote a set containing an infinite sequence of complex numbers. For example,

\[x = [x_1, x_2, x_3 \ldots] \quad \text{and} \quad y = [y_1, y_2, y_3 \ldots] \quad \text{are 2 elements of } V.
\]
Assume \( V \) also contains \( 0 = [0,0,0,\ldots] \).

On \( V \) we define the 2 operations:

\[
\begin{align*}
x + y &= [x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots] \\
\lambda x &= [\lambda x_1, \lambda x_2, \ldots]
\end{align*}
\]

compactly:

\[
\begin{align*}
(x + y)_n &= x_n + y_n \\
(\lambda x)_n &= \lambda x_n.
\end{align*}
\]

The elements of \( V \) form a vector space, since the operations of addition and scaling apply, and \( V \) has a notion of zero. It is an infinite-dimensional vector space. It is spanned by the vectors

\[
\begin{align*}
V^1 &= [1,0,\ldots] \\
V^2 &= [0,1,0,\ldots] \\
V^3 &= [0,0,1,\ldots] \\
&\vdots
\end{align*}
\]

definition: Linear operators map elements of the vector space to itself:

\[
L : V \rightarrow V.
\]

Example The displacement operator \( E \) is a linear operator:

let \( x = [x_1,x_2,x_3,\ldots] \)

Define \( Ex := [x_2,x_3,x_4,\ldots] \)

\[
\begin{align*}
(Ex)_n &= x_{n+1} \\
(EEx)_n &= (Ex)_{n+1} = x_{n+2} \\
(E^kx)_n &= x_{n+k}
\end{align*}
\]

Example The linear difference operator

\[
L = \sum_{i=0}^{m} c_i E^i
\]

(112)
(with constant coefficients $c_i$ and finite rank) is also a linear operator:

$$E^0 = \text{identity operator} \ (E^0 x)_n = x_n$$

$L$ is a polynomial in $E$ and can be written like so in powers of $E$:

$$L = p(E)$$

$$p(\lambda) = \sum_{i=0}^{m} c_i \lambda^i$$
is the characteristic polynomial.

14.1 Homogeneous Difference Equations

Want to solve

$$Lx = 0,$$

where $L$ is given by (??). The set of solutions $\{x : Lx = 0\}$ is a linear subspace of $V$. It is called the null space. In fact, $Lx = 0$ is solved if we know the basis for the null space of $L$.

Let’s learn how to solve these constant coefficient homogeneous difference equations by example: Suppose $c_0 = 2 \ c_1 = -3 \ c_2 = 1$. Then

$$L = E^2 - 3 E^1 + 2 E^0,$$

$$L x = (E^2 - 3 E^1 + 2 E^0)x = 0,$$

$$x_{n+2} - 3x_{n+1} + 2x_n = 0 \quad n \geq 0.$$

Associated with this problem with characteristic polynomial

$$p(\lambda) = \sum_{i=0}^{2} c_i \lambda^i = \lambda^2 - 3\lambda + 2,$$

is a characteristic equation,

$$p(E)x = 0.$$

The trivial solution is always a solution, i.e., $x = 0$. But there are other solutions: We guess that $x_n = \lambda^n$. Substituting this guess in the characteristic equation yields

$$\lambda^n - 3\lambda^{n+1} + 2\lambda^n = 0,$$
or

\[ \lambda^n(\lambda^2 - 3\lambda + 2) = 0. \]

Rewriting,

\[ \lambda^n(\lambda - 2)(\lambda - 1) = 0, \]

has solutions \([0, 0, 0 \ldots]\), \(v_n = 2^n\) and \(u_n = 1^n\). The first one is the trivial solution, the other two are \textit{simple} zeros of the polynomial.

It turns out that \(u = u_n\) \(v = v_n\) form a basis: To see this, form a linear combination of the form \(x = \alpha u + \beta v\). Specifically, let

\[ x_n = \alpha u_n + \beta v_n \quad \forall n. \]

(113) if \(n = 1\) then \(\begin{cases} x_1 = \alpha + 2\beta, \\ x_2 = \alpha + 4\beta. \end{cases}\)

(113) determines \(\alpha, \beta\) since \(\det \begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} \neq 0\) (This is a Vandermonde Matrix).

Let’s check:

\[
\begin{align*}
x_n &= 3x_{n-1} - 2x_{n-2}, \\
&= 3(\alpha u_{n-1} + \beta u_{n-1}) - 2(\alpha u_{n-2} - \beta v_{n-2}), \\
&= \alpha(3u_{n-1} - 2u_{n-2}) + \beta(3v_{n-1} - 2v_{n-2}), \\
&= \alpha u_n + \beta v_n.
\end{align*}
\]

The guess we used for \(x_n\) is appropriate when the characteristic equation has simple zeros.

**Theorem:** If \(p\) is the characteristic polynomial and \(\lambda\) is a zero of \(p \Rightarrow\) one solution of \(p(E)x = 0\) is \([\lambda, \lambda^2, \lambda^3 \ldots]\). If all zeros of \(p\) are simple and nonzero then a solution of the difference equation is a linear combination of such special solutions.

\[ \square \]

What to do when \(p\) has non-simple zeros?
Zeros of Higher Multiplicity: Let’s guess that
\[ x(\lambda) = [\lambda, \lambda^2, \lambda^3 \ldots]. \]

Then
\[ p(E)x(\lambda) = p(\lambda)x(\lambda). \]

Next, differentiate with respect to \( \lambda \):
\[ p(E)x'(\lambda) = p'(\lambda)x(\lambda) + p(\lambda)x'(\lambda). \]

If \( \lambda \) is a multiple zero of \( p \), then \( p(\lambda) = p'(\lambda) = 0 \). Therefore \( x(\lambda) \) and \( x'(\lambda) \) solve the difference equation.

Take \( x'(\lambda) = [1, 2\lambda, 3\lambda^2, \cdots] \). If \( \lambda \neq 0 \), it’s independent of the solution \( x(\lambda) \) because \( \det \begin{bmatrix} \lambda & \lambda^2 \\ 1 & 2\lambda \end{bmatrix} \neq 0 \).

Therefore, if the sequence is truncated at the second term the resulting vectors in \( \mathbb{R}^2 \) are linearly independent.

Using the same reasoning we can show that if \( \lambda \) is a zero of \( p \) having \( k \) multiplicity, then
\[ p(E)x = 0, \]

has solutions
\[ x(\lambda) = [\lambda, \lambda^2, \lambda^3 \ldots], \]
\[ x'(\lambda) = [1, 2\lambda, 3\lambda^2 \ldots], \]
\[ x''(\lambda) = [0, 2, 6\lambda \cdots], \]
\[ \vdots \]
\[ x^{k-1}(\lambda) = \frac{d^{k-1}}{d\lambda^{k-1}}[\lambda, \lambda^2, \lambda^3 \ldots]. \]

Theorem Let \( p \) be the characteristic polynomial with \( p(0) \neq 0 \). Then the basis of the null space of \( p(E) \) is obtained as follows: for each zero of \( p \) having multiplicity \( k \) associate the \( k \) basic solution \( x(\lambda), x'(\lambda), x''(\lambda) \cdots x^{k-1}(\lambda) \)

where \( (\lambda) = [\lambda, \lambda^2, \lambda^3 \ldots] \),

297
i.e., general solution if \( \lambda_0 \) is one such non-simple zeros, the the general solution is
\[
\alpha_1 x(\lambda_0) + \alpha_2 x'(\lambda_0) + \ldots \alpha_{k-1} x^{k-1}(\lambda_0).
\]

**Example**

\[
4x_n - 9x_{n-1} + 6x_{n-2} - x_{n-3} = 0
\]

let \( x_n = \lambda^n \Rightarrow \lambda^n(4 - 9\lambda^{-1} + 6\lambda^{-2} - \lambda^{-3}) = 0 \)
\[
\lambda^n(4\lambda^3 - 9\lambda^2 + 6\lambda - 1) = 0
\]
\[
\lambda^n(\lambda + 1)^2(4\lambda - 1) = 0
\]

The zeros of the characteristic are: \(-1\) (double, \( k = 2 \)), \(1/4\) (simple).

Let
\[
x(\lambda) = [\lambda, \lambda^2, \lambda^3 \ldots],
\]

then
\[
x'(\lambda) = [1, 2\lambda, 3\lambda^2 \ldots].
\]

Evaluating these explicitly,
\[
x(-1) = [-1, 1, -1 \ldots], \quad x\left(\frac{1}{4}\right) = \left[\frac{1}{4}, \frac{1}{4^2}, \frac{1}{4^3}, \ldots\right]
\]
\[
x'(-1) = [1, -2, 3, -4 \ldots].
\]

The general solution is then
\[
x = \alpha x(-1) + \beta x'(-1) + \lambda x\left(\frac{1}{4}\right),
\]
or
\[
x_n = \alpha(-1)^n + \beta n(-1)^{n-1} + \lambda \left(\frac{1}{4}\right)^n.
\]

Solving non-constant coefficient linear difference equations is more complicated. Variation of parameters or a lucky guess might do the trick. In fact, it never hurts to try to guess a solution: consider the following example.

**Example** Consider \( x_{n+1} - nx_n = 0 \), not polynomial.

By inspection:
\[ n = 1 \quad x_2 = x_1. \]
\[ n = 2 \quad x_3 = 2x_2 = 2x_1, \]
\[ n = 3 \quad x_4 = 3x_3 = 3 \cdot 2x_1. \]

So the general solution is
\[ x_n = (n - 1)!x_1, \quad n > 1. \]

\[ \Box \]

### 14.2 Stable Difference Equations

Let \( x = [x_1, x_2 \cdots x_n \cdots] \in V \), a linear vector space. We say that \( x \) is bounded if \( \exists c \) constant such that \( |x_n| \leq c \forall n \). That is,
\[ \sup_n |x_n| < \infty. \]

The linear difference equation \( p(E)x = 0 \) is stable if all its solutions are bounded.

**Theorem** For \( p \) with \( p(0) \neq 0 \), the following statements are equivalent:
\[
\begin{cases}
(\text{i}) \quad p(E)x = 0 \text{ is stable.} \\
(\text{ii}) \quad \text{All zeros } z \text{ of } p \text{ satisfy } |z| \leq 1 \quad \text{if simple zeros.} \\
\quad \text{They must satisfy } |z| < 1 \text{ if zeros of higher multiplicity.}
\end{cases}
\]

**Example** Consider \( 4x_n + 7\lambda x_{n-1} + 2x_{n-2} - x_{n-3} = 0 \). Are the solutions stable?
The characteristic equation is \( 4\lambda^3 + 7\lambda^2 + 2\lambda - 1 = 0 \). Thus \( p \) has a zero at \(-1\), which is double and a zero \( \frac{1}{4} \). Thus the solutions are unstable.

\[ \Box \]

**Example:** Consider \( x_n = x_{n-1} + x_{n-2} \quad \lambda^2 - \lambda + 4 = 0 \) roots:
\[ (1 \pm \sqrt{5}) / 2. \] The solutions are unstable.

\[ \Box \]