INTRODUCTION TO SCIENTIFIC COMPUTING –DRAFT

July, 2001

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)

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 $f(x) = x$. 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

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

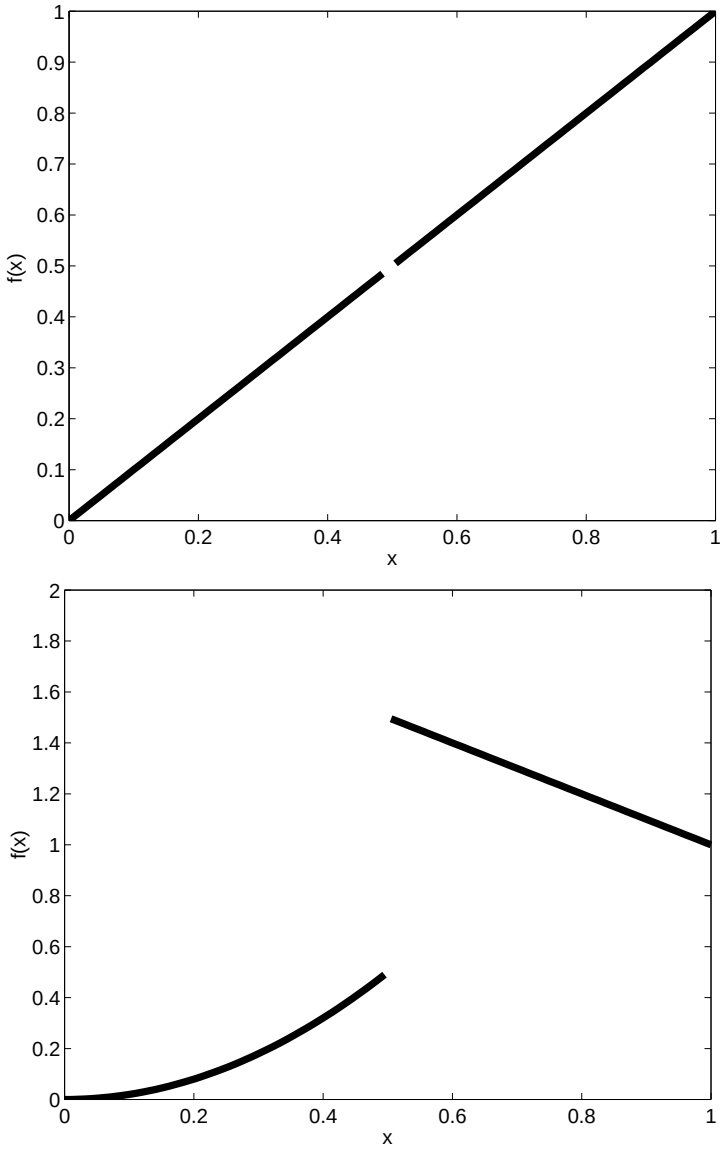

Figure 1: Examples of (a) a removable discontinuity and (b) a jump discontinuity.

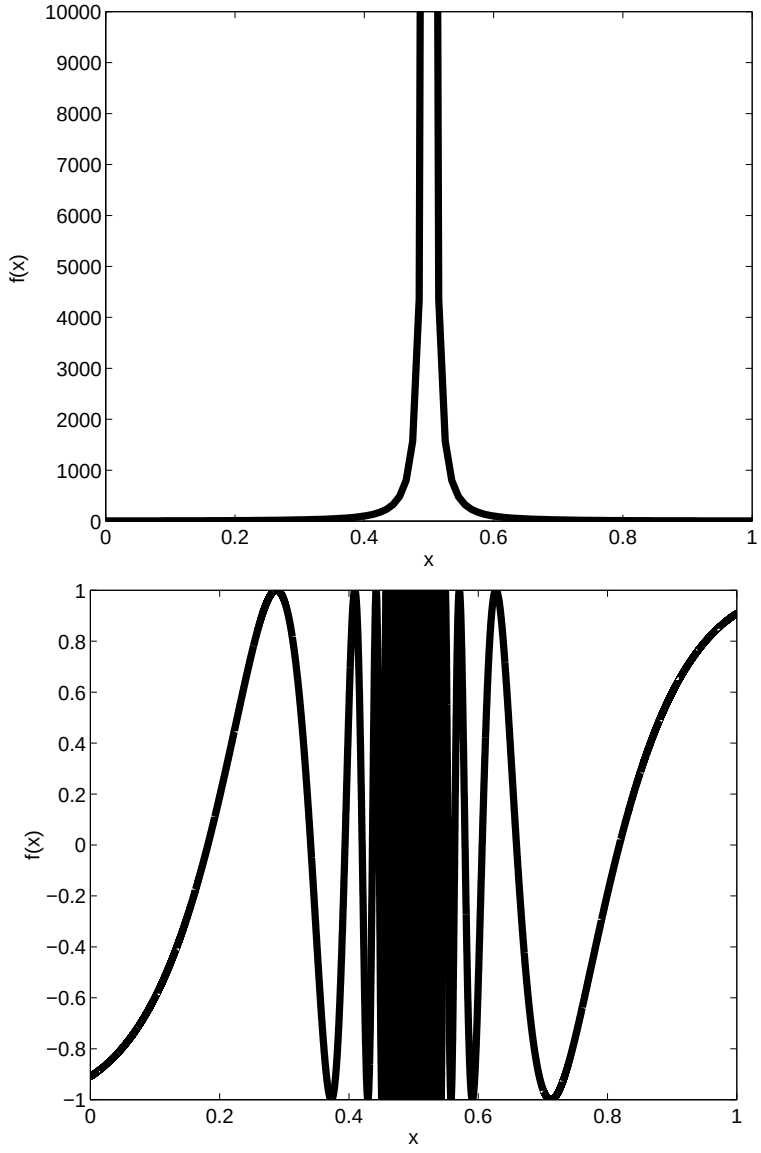

Figure 2: Two more types of discontinuities

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

$$f(x) = \sin(\frac{1}{x - 0.5}).$$

As $x \to 0.5$ this function oscillates more and more wildly between ± 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

$$f(x) = \begin{cases} 2x^2 & x \leq 0.5 \\ \frac{x}{4} + 0.375 & x > 0.5 \end{cases}.$$

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 $C^2([0, 1])$.

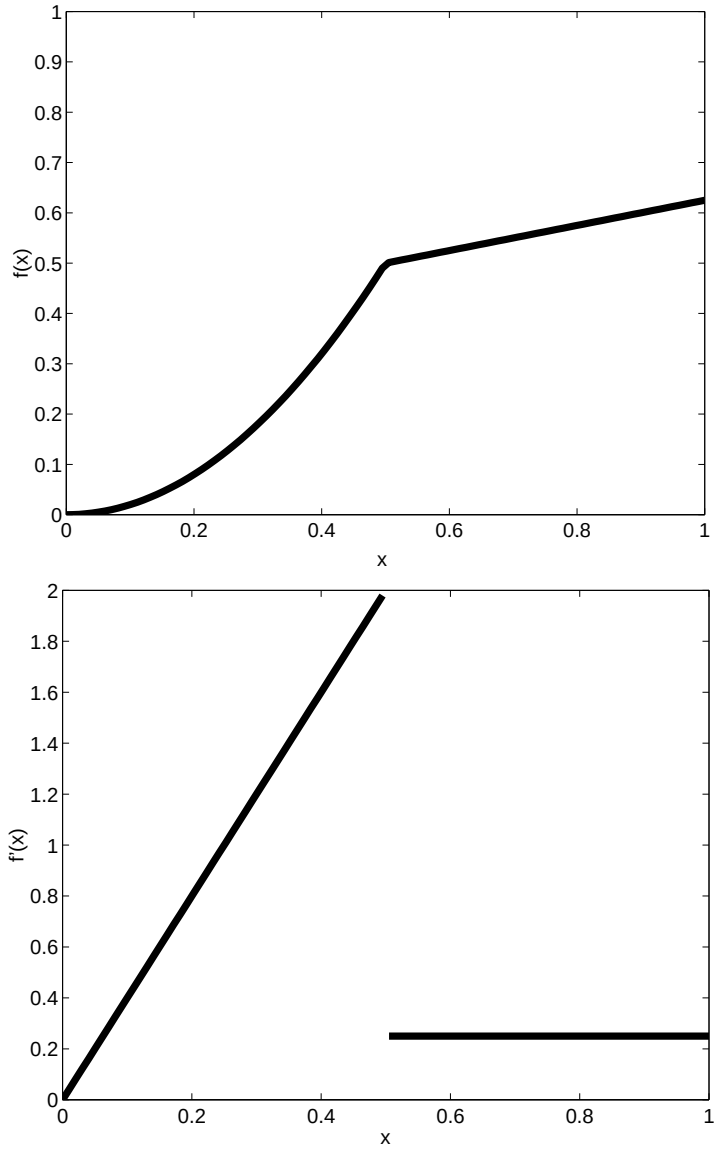

Figure 3: A function that is $C([0,1])$ but not $C^1([0,1])$. We can see that although the function is continuous it is not 'smooth' because there is a sharp bend at $x = 0.5$. When we look at the derivative of the function we can see that it is not continuous because there is a jump in the derivative at $x = 0.5$.

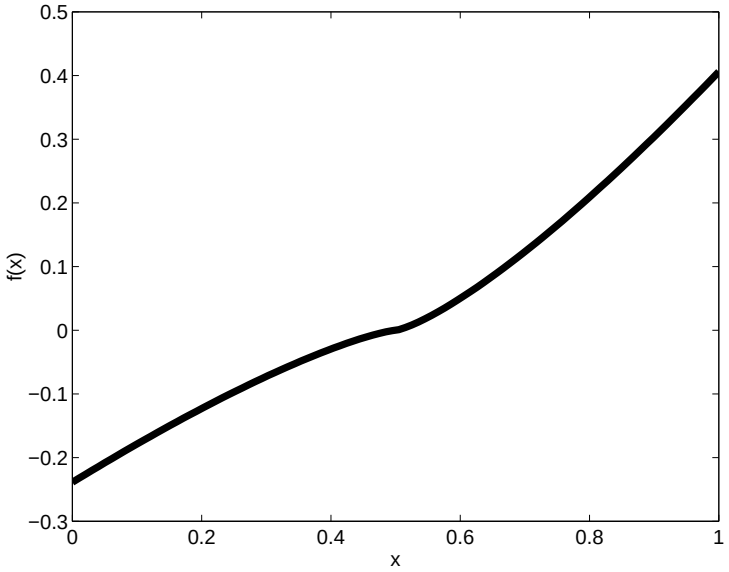

Figure 4: A $C^1([0,1])$ function but not a $C^2([0,1])$ function, $f(x) = (x - 0.5)^{1.3}$. Figure 5 shows the first two derivatives of $f(x)$.

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. 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

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

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 $f(c) = k$

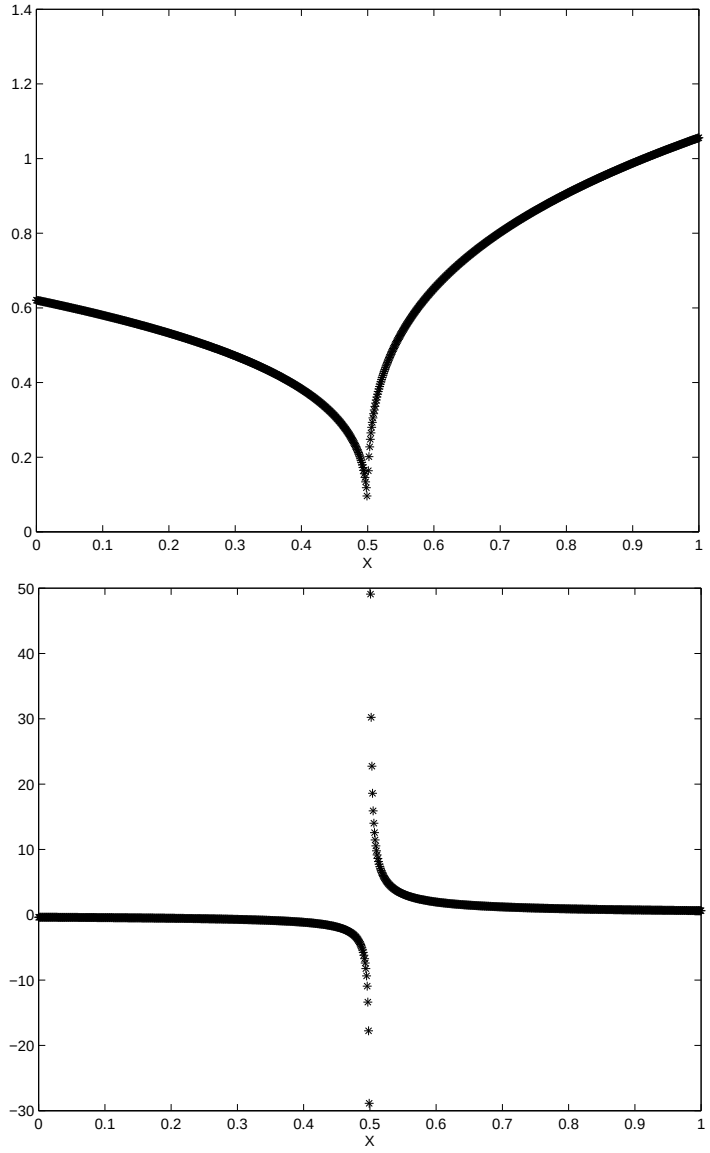

Figure 5: The first and second derivatives of $f(x) = (x - 0.5)^{1.3}$. The first derivative of $f(x)$ is continuous so $f(x) \in C^1([0, 1])$ but we can see a sharp point at $x = 0.5$. This becomes a discontinuity in the second derivative so $f(x)$ is not in $C^2([0, 1])$.

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

$$w_1 q(x) \leq w(x)q(x) \leq w_2 q(x), \quad w_1 M \leq \int_a^b q(x)w(x)dx \leq w_2 M,$$

where $M := \int_a^b q(x)dx$, and the intermediate value theorem for functions. That is,

$$w(s)M = \int_a^b w(x)q(x)dx.$$

□

1.3 Orders of Convergence

Convergent Sequences: suppose you are solving an integral by an iterative technique and the code produces a sequence of real numbers $x_1, x_2, x_3 \cdots$ tending toward a number.

We write $\lim_{n\to\infty} x_n = L$,

if there corresponds to each $\varepsilon > 0$ a real number r such that $|x_n - L| < \varepsilon$ whenever $n > r$ (here n integer).

Example: you can confirm, by numerical experimentation (i.e., make a table) that $\lim_{n\to\infty} \dfrac{n+1}{n} = 1$. We can see that $|\frac{n+1}{n} - 1| < \varepsilon$ whenever $n > \dfrac{1}{\varepsilon}$ □

Example Take $x_n = \left(1 + \frac{1}{n}\right)^n$, $n = 1, 2, ...$ Numerical calculation shows that:

$$\begin{aligned} x_1 &= 2.0 \\ x_{10} &= 2.593742 \\ x_{30} &= 2.674319 \\ x_{50} &= 2.691588 \\ x_{1000} &= 2.716924 \\ &\text{etc.} \end{aligned}$$

You can also confirm numerically that

$$\left| \frac{x_{n+1} - e}{x_n - e} \right| \to 1.$$

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 worse than linear convergence.

□

Example: Consider $x_{n+1} = x_n - \dfrac{x_n^2}{x_n^2 + x_{n-1}^2}$, $n = 0, 1, 2, ...,$

$$\begin{aligned} x_0 &= 20.0 \\ x_1 &= 15.0 \\ x_2 &= 14.64 \\ x_{33} &= 0.54 \\ x_{34} &= 0.27 \end{aligned}$$

This one has more rapid convergence than the previous case. In fact, numerical 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,

$$\begin{aligned} x_1 &= 2 \\ x_2 &= 1.5 \\ x_3 &= 1.416667 \\ x_4 &= 1.41421 \\ &\text{etc.} \end{aligned}$$

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...the ratio is a constant.

□

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:

$$|x_{n+1} - x^*| \leq \varepsilon_n|x_n - x^*| \quad n \geq N$$

and $\varepsilon_n \to 0$.

Quadratic:

$$|x_{n+1} - x^*| \leq K|x_n - x^*|^2, \text{have } K \text{ not necessarily less than 1}$$

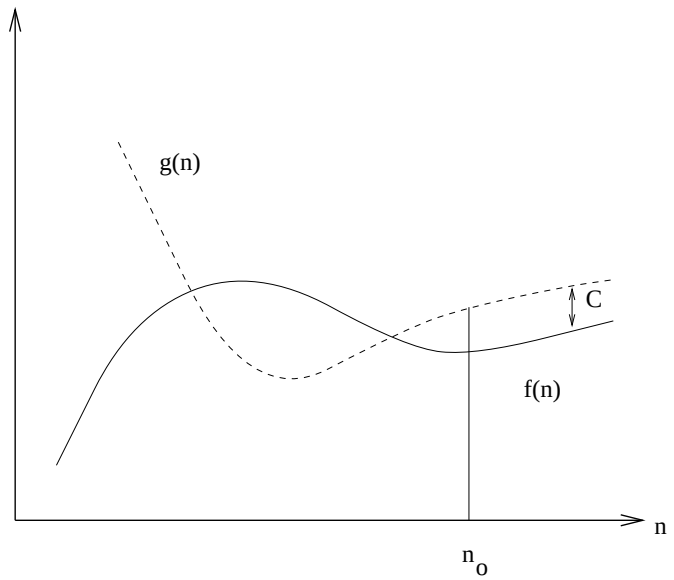

Figure 6: $f(n) = \mathcal{O}(g(n))$ as $n \to \infty$. Here $g(n) \neq 0$.

General case: If

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

we say "at least α-order convergent."

□

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

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

Write $\quad x_n = \mathcal{O}(\alpha_n)$,

if there exist K and n_0 constants, such that $|x_n| \leq K|\alpha_n|$ when $n \geq n_0$.

Big $\mathcal{O}$ gives an upper bound on x_n, to within a constant.

Example Figure 6 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

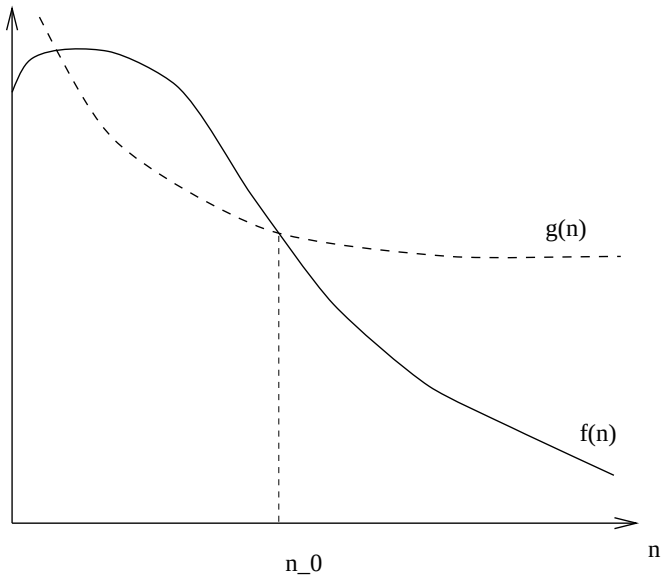

Figure 7: $f(n)$ becomes insignificant with respect to $g(n)$ as $n \to \infty$.

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 $n \geq n_0$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

i.e. $f(n)$ becomes insignificant with respect to $g(n)$ as $n \to \infty$, see Figure 7.

Figure 7 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 α_n. If $x_n = o(\alpha_n)$ then x_n converges to "more" than α_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,

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

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

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

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

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

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

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

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

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

□

Example: Consider the following functions

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

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

$$\begin{aligned}\lim_{n\to\infty} \frac{h(n)}{g(n)} &= \lim_{n\to\infty} \frac{1+4e^{2n}}{e^{2n}} \\ &= \lim_{n\to\infty} \frac{1}{e^{2n}} + 4 \\ &= 4\end{aligned}$$

□

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

$$\begin{aligned}\lim_{n\to\infty} \frac{k(n)}{f(n)} &= \lim_{n\to\infty} \frac{5-e^n+n}{e^n} \\ &= \lim_{n\to\infty} \frac{5}{e^n} - 1 + \frac{n}{e^n} \\ &= -1.\end{aligned}$$

□

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

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

□

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{e^n}{e^{2n}}$$

$$\begin{aligned} &= \lim_{n\to\infty} \frac{1}{e^n} \\ &= 0. \end{aligned}$$

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

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

□

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

$$\begin{aligned} f(\epsilon) &= 1 + 4\epsilon^2 \\ g(\epsilon) &= \sqrt{\epsilon} + 3\epsilon^3 \\ h(\epsilon) &= \epsilon(1 - \sqrt{\epsilon}) \end{aligned}$$

We can see that $h(\epsilon) = o(g(\epsilon))$ (remember that we are now interested in $\epsilon \to 0$)

$$\begin{aligned} \lim_{\epsilon\to 0} \frac{h(\epsilon)}{g(\epsilon)} &= \lim_{\epsilon\to 0} \frac{\epsilon(1-\sqrt{\epsilon})}{\sqrt{\epsilon} + 3\epsilon^3} \\ = \lim_{\epsilon\to 0} \frac{\epsilon(1-\sqrt{\epsilon})}{\sqrt{\epsilon}(1+3\epsilon^{\frac{5}{2}})} & \\ &= \frac{\sqrt{\epsilon}(1-\sqrt{\epsilon})}{(1+3\epsilon^{\frac{5}{2}})} \\ &= 0 \end{aligned}$$

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

$$\begin{aligned} \lim_{\epsilon\to 0} \frac{g(\epsilon)}{f(\epsilon)} &= \lim_{\epsilon\to 0} \frac{\sqrt{\epsilon} + 3\epsilon^3}{1 + 4\epsilon^2} \\ &= \lim_{\epsilon\to 0} \frac{\sqrt{\epsilon}(1+3\epsilon^{\frac{5}{2}})}{1 + 4\epsilon^2} \\ &= 0 \end{aligned}$$

Now let us compare $h(\epsilon)$ and $f(\epsilon)$

$$\begin{aligned}\lim_{\epsilon\to 0}\frac{h(\epsilon)}{f(\epsilon)} &= \lim_{\epsilon\to 0}\frac{\epsilon(1-\sqrt{\epsilon})}{1+4\epsilon^2}\\ &= 0.\end{aligned}$$

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.

$$\begin{aligned}&O(o(f)) = o(O(f)) = o(o(f)) = o(f)\\ &O(fg) = O(f)O(g)\\ &O(f) + o(f) = O(f)\\ &O(O(f)) = O(f)\\ &O(f) - O(f) = O(f)\\ &O(f) + O(f) = O(f).\end{aligned}$$

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 under-

stand 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:

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} a_n (x - x_0)^n = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 + \frac{1}{3!} f'''(x_0)(x - x_0)^3 ... \\ &+ \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n + ... \end{aligned}$$

(if $x_0 = 0$ the series is called a Maclaurin series). Clearly,

$$a_n = \frac{f^{(n)}(x_0)}{n!}, \quad n = 0, 1, 2, ...$$

If a function depends on 2 variables x, y, say, then

$$\begin{aligned} f(x, y) &= f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0) \\ &+ \frac{1}{2!}[f_{xx}(x_0, y_0)(x - x_0)^2 + 2 f_{xy}(x_0, y_0)(x - x_0)(y - y_0) + f_{yy}(x_0, y_0)(y - y_0)^2] + ... \end{aligned}$$

about x_0 and y_0. Here, $f_x := \partial f / \partial x$, 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 $f : (R) \to \mathbb{R}$ 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

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 ... \\ &+ \frac{1}{k!}f^{(k)}(x_0)(x - x_0)^k + h_k(x)(x - x_0)^k, \end{aligned}$$

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 ξ 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 *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 abbacus, 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:

$$\begin{aligned} \beta &\text{ base or radix} \\ t &\text{ Precision} \\ [L, U] &\text{ exponent range} \end{aligned}$$

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 β representation:

$$x = \pm \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} \cdots \frac{d_t}{\beta^t} \right) \beta^e = \pm \underbrace{0.d_1 d_2 d_3 \cdots d_t \times \beta^e}_{\text{normalized representation}}$$

$$\begin{aligned} \beta &\text{ base or radix} \\ t &\text{ Precision} \\ [L, U] &\text{ exponent range} \end{aligned}$$

$$\text{where } 0 \leq d_i \leq \beta - 1, \quad i = 0, \ldots, t$$

$$L \leq e \leq U$$

So then the string of base β represented by $\overbrace{d_1 d_2 \cdots d_t}$ $\equiv$ mantissa or significand and $e \equiv$ exponent or characteristic The *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 8, for example.

Exercise: Show that the total number of machine numbers, for a machine operating in base β 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: $\beta = 16 \qquad 0 \leq d_i \leq 15$ where

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

Example

$$\begin{aligned}(23A.D)_{16} &= 2 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 + 13 \times 16^{-1} = 512 + 48 + 10 + 0.8125 \\ &= 570.8125\end{aligned}$$

□

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.

$$\begin{aligned}(1101.0011)_2 &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= 8 + 4 + 0 + 1 + 0 + 0 + 0.125 + 0.0625 \\ &= 13.1875\end{aligned}$$

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

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

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

$$
\begin{aligned}
&\frac{2}{3} = (0.a_1a_2a_3\cdots)_2 \\
&\times 2 \\
&\frac{4}{3} = (a_1.a_2a_3\cdots)_2 \Rightarrow a_2 = 1, \text{by taking the integer part of b.sides} \\
&\frac{4}{3} - 1 = \frac{1}{3} = (0.a_2a_3\cdots)_2 \\
&\times 2 \\
&\frac{2}{3} = (a_2.a_3a_4\cdots)_2 \\
&\times 2 \\
&\frac{4}{3} = (a_2a_3.a_4\cdots)_2 \ \therefore a_2 = 0, a_3 = 1 \\
&\frac{4}{3} - 1 = \frac{1}{3} = (0.a_4a_5\cdots)_2 \\
&\times 2 \\
&\frac{2}{3} = (a_4.a_5\cdots)_2 \\
&\times 2 \\
&\frac{4}{3} = (a_4a_5.a_6\cdots)_2 \therefore a_4 = 0 \quad a_5 = 1 \\
&\frac{2}{3} = (0.10101\cdots) = (1.0101\cdots)\cdot 2^{-1}
\end{aligned}
$$

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

Standard for vendors

System	β	t	L	U
IEEE HP, binary 16 bit word	2	11	-14	15
IEEE SP, binary 32 bit word	2	24	-126	127
IEEE DP, binary 64 bit word	2	53	-1022	1023
IEEE QP, binary 128 bit word	2	113	-16382	16383
Cray	2	48	-16383	16384
HP Calculator	10	12	-499	499
IBM 3000	16	6	-64	63

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 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 β^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 8 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 $0 < e < (11 \quad 111 \quad 111)_2 =$

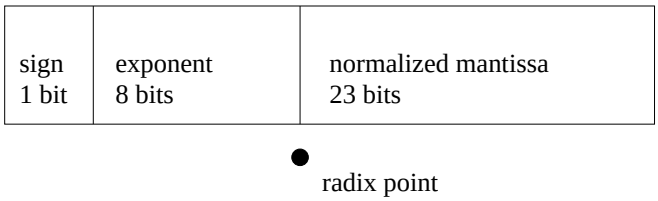

Figure 8: *Schematic representation of the storage on a machine that uses 32-bit word length. Here the mantissa is 24-bit in length, where the first bit* $= 1$ *(is implied, and we're going to use the convention that the first bit is an implied 1). Each section is called a field -sometimes the sign bit is assigned to the exponent field-.*

$2^8 - 1 = 255$; the number 0 is reserved for ± 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
$fp(\widehat{x}) = 0.47584 \cdot 10^3$	chopping
$fp(\widehat{x}) = 0.47585 \cdot 10^3$	rounding.
Take	
$x = 0.333333 \cdots$	
$fp(\widehat{x}) = 0.33333$	chopping
$fp(\widehat{x}) = 0.33333$	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 $0.10101010\ldots10$ and $0.101010\ldots11$. 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[2]

[2]Gradual underflow preserves more mathematical identities. For example, $x - y = 0$

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:

$$\begin{aligned} x_- &= (1.0101\cdots010)\cdot 2^{-1} \leftarrow \text{chopping} \\ x_+ &= \underbrace{(1.0101\cdots011)}_{24bits}\cdot 2^{-1} \leftarrow \text{rounding up} \end{aligned}$$

Which $\begin{cases} x_+ \text{do we take for } \widehat{x} \text{ of } \dfrac{2}{3}? \\ x_- \end{cases}$

Choose the closest:

$$x - x_- = (0.1010\cdots)_2 \cdot 2^{-24} = \frac{2}{3}\cdot 2^{-24}$$

$$x_+ - x = (x_+ - x_-) + (x_- - x) = \frac{1}{3}\cdot 2^{-24}$$

$$\therefore \text{take } \widehat{x} = x_+ . \text{The absolute roundoff?}$$

$$|\widehat{x} - x| = \frac{1}{3}\cdot 2^{-24}$$

$$\text{relative roundoff}$$

$$\frac{|\widehat{x} - x|}{|x|} = \frac{\frac{1}{3}\cdot 2^{-24}}{\frac{2}{3}} = 2^{-25}$$

2.2 Machine Precision:

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

$$\varepsilon_{mach} = \beta^{1-t} \quad \text{for chopping}$$

$$\varepsilon_{mach} = \frac{1}{2}\beta^{1-t} \quad \text{for rounding}$$

imples $x = y$ 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.

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

$$\frac{|fp(x) - x|}{|x|} \leq \varepsilon_{mach}$$

Not to be confused with UFL : $0 < UFL < \varepsilon_{mach} < OFL$, i.e. with underflow and overflow.

Example For IEEE single precision: $\varepsilon_{mach} = 2^{-24} \approx 10^{-7}$ "7 decimal digit precision"
double precision: $\varepsilon_{mach} = 2^{-53} \approx 10^{-16}$ sixteen decimal precision"

For chopping: let $fp(x) = x(1 \pm \varepsilon)$, $\varepsilon(x)$ in general

Assume (+):

$$\begin{aligned} |x - fp(x)| &= (0.0 \cdots 0 d_t d_{t+1} d_{t+2} \cdots)_\beta \times \beta^e \\ &= (0 \cdot d_t d_{t+1} d_{t+2} \cdots)_\beta \times \beta^{e-(t-1)} \leq \beta^{e-(t-1)} \end{aligned}$$

$$\frac{(0.d_t d_{t+1}, \cdots)\beta^{e-(t-1)}}{(0.d_1 \cdots)\beta^e} = \frac{(0.d_t d_{t+1}, \cdots)}{(0.d_1 \cdots)} \beta^{-(t-1)} \leq \beta^{-(t-1)}$$

$$\frac{|x - x(1+\varepsilon)|}{|x|} = \frac{|x\varepsilon|}{|x|} = |\varepsilon| \leq \beta^{-(t-1)}$$

exercise: Work out the rounding case. □

Some examples: ε_{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.

We say that $\hat{x}$ has m significant digits with respect to x if m is the largest nonnegative integer which

$$\frac{|x - \widehat{x}|}{|x|} \leq \frac{1}{2}\beta^{1-m}.$$

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

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

$$\frac{|x - \widehat{x}|}{|x|} = \frac{0.00011}{002136} \approx 0.00515 \leq 5 \cdot 10^{-2}$$

Thus $\widehat{x}$ 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?

[3] A useful theorem for *binary* numbers:

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 $x - y$

□

Let the symbol * represent $\{+, -, /, .\}$, 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:

$$(x * y) - fp(\widehat{x} * \widehat{y}) = \underbrace{(x * y - \widehat{x} * \widehat{y})}_{\text{propagated error}} + \underbrace{(\widehat{x} * \widehat{y} - fp(\widehat{x} * \widehat{y})}_{\text{round off}}$$

□

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 $\widehat{x} = 4.34$ and $\widehat{y} = 1.02$, using 3-digit rounding.

The product of these, to 3-digit rounding: $fp(\widehat{x} \times \widehat{y}) = fp(4.34 \times 1.02) = fp(4.4268) = 4.43$. 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 $4.335 \leq x \leq 4.345 \quad 1.015 \leq y \leq 1.025$. Thus, $4.400025 \leq x \times y \leq 4.556125$. The propagated error is thus $-0.129325 \leq x \times y - \widehat{x} \times \widehat{y} \leq 0.026775$
□

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 $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

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,

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

$$b < 0$$

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

  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 $a = 0.05010 \quad b = -98.78 \quad c = 5.015$. 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:

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

Using the other formula

$$\frac{10.03}{98.78 \pm 98.77} = \begin{cases} 1003 \leftarrow \text{large error} \\ 0.05077 \end{cases}$$

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 $\{P\}_{i=1}^{N}$ and $\{Q\}_{i=1}^{N}$, 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!

   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. At this point you can focus on computer issues, and with a matlab code, you have a benchmark to which the new code can be checked against.

   I consider all programming languages equally terrible, or equally good: the key is using 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.

2) Decide on the algorithm and the extent of the how robust you want to make the code based on the following criteria: (a) is this a one-shot code? or (b) is this a production code? Why this decision is very important is this: you want to get to results as quickly as possible. You factor in the amount of time it will take to compute the answer, but also the time it takes you to design a code. For example, if you work on a code for a year just to maximize the speed of a code you should remember that the code is now one year old by the time you use it and it might be better to spend only one month on the code and already have 11 months in the balance that you can use to run

the code. In these cases, one-shot code is preferred. On the other hand, a production code is preferred if the code will be used a great many times, or by people other than yourself. In this case, software engineering should be geared to increase its efficiency and/or to make it robust so that others can use it with minimal problems.

a) In the case of a one-shot code, you want to provide accurate results. It will be used to get results quickly and then the code is no longer of use...if your programming is very modular, however, each module can be stored for later use in your "Tool-kit" should you need it at some later time.

b) Software design is crucial $\rightarrow$ take the time to optimize both the operation and the scheme.

3) Make estimates of computing and storage requirements as early as possible.

4) Write code on paper and do an analytical estimation of its properties.

5) Implement using matlab (forces you to keep problem small). As I said, the other nice thing you'll get from the matlab code is a good test to benchmark your bigger codes once they are built.

6) Do not reinvent the wheel $\rightarrow$ use (good) developed code, when available. There's tons of this stuff out there. Mostly developed by specialists in each particular algorithm or mathematical area.

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

8) 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. You might want to have a directory (which I call the sandbox) in which I dump all the old versions. I do a sandbox cleanup periodically. I also carry a simple "diary" in which I enter comments on the code's progress. This is especially helpful if I have to leave a particular project for an extended period of time and then want to remember as quickly as possible where I left off.

9) 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.

10) 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.

11) Use descriptive names for variables and don't make them single letters...these are a pain to find using an editor.

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

10) Write README's with usage instructions, changes, bugs, etc.

11) When working with others on a code, a useful thing to do is for there be, at any time, a "keeper of the code". You declare yourself this person and your code and only your version is the official one...all others will defacto have now "old" codes, regardless of whether your codes happen to be identical.You are only allowed to make changes on the code if you are the keeper of the code and must "release" the code to the new keeper, if this other person will make changes. By the way, there are commercial and free-ware software packages that perform this routine (and keeps records).

12) Test, test, test your code.

3.2 The Big Picture on Numerical Schemes

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.

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:

let ε= initial error, then, after n steps:

$$|E_n(\varepsilon)| \approx C\, n\, \varepsilon \quad \text{linear growth in } n \text{ is acceptable.}$$
$$|E_n(\varepsilon)| \approx k^n \varepsilon \quad k > 1 \quad \text{exponential growth is not acceptable.}$$

See Figure 9.

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.

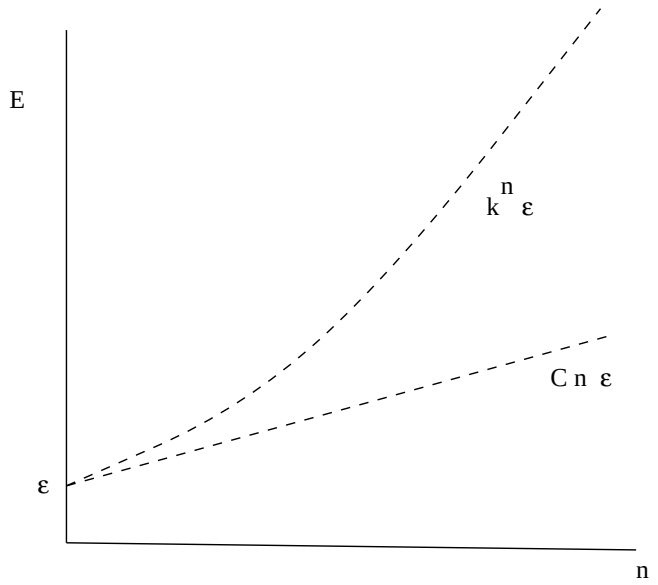

Figure 9: linear growth is fine: if you perturb a stable system you expect at most a linearly proportional response to the bump, but not anything polynomial or exponential.

Example Solving a very large $Ax = b$ can be solved by direct methods, or iterative methods, as we shall see. In general an iterative method owes its existence to the unacceptable cost in either time or computer resources.

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 $\begin{cases} \text{Approximation: replace a complicated process by a simpler process} \\ \text{Truncation or discretization.} \\ \text{Rounding: computer representation of real numbers} \\ \text{and arithmetic operators.} \end{cases}$

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 abolute difference between the estimator device and the true population 10^3, does this mean that the estimator is an innaccurate 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.

□

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.

$$\begin{cases} \text{Stability: used to categorize problems and algorithms} \\ \text{Conditioning: categorizes algorithms and methods} \end{cases}$$

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 $\widehat{x}$ (assume $\widehat{x}$ close to x and within the range of values for which $f(x)$ has the above continuity)

$$f(x) = f(\widehat{x}) + \frac{df(\xi)}{dx}(x - \widehat{x}) \quad \text{by the mean-value-theorem}$$
$$x \leq \xi \leq \widehat{x}$$

Abs $(f(\widehat{x})) = |f(x) - f(\widehat{x})| = f'(\xi)|x - \widehat{x}| \approx f'(\widehat{x}) \cdot$ Abs $(\widehat{x})$

Example $f(x) = \sqrt{x} \quad \therefore f'(x) = \dfrac{1}{2\sqrt{x}} \therefore$

Abs $(\sqrt{\widehat{x}}) \approx \dfrac{Abs(\widehat{x})}{2\sqrt{\widehat{x}}}$. For relative error:

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

Hence, since the relative error in $\sqrt{\widehat{x}} \sim \frac{1}{2}$ relative error in $\widehat{x}$ thus $\sqrt{\ }$ is a safe operation.

□

Suppose result depends on more than one variable, say x, y?

$$f(x,y) = f(\widehat{x},\widehat{y}) + \frac{\partial f}{\partial x}(\widehat{x},\widehat{y})(x-\widehat{x}) + \frac{\partial f}{\partial y}(\widehat{x},\widehat{y})(y-\widehat{y})$$
$$\therefore \text{Abs}(f(\widehat{x},\widehat{y})) \approx |\frac{\partial f}{\partial x}(\widehat{x},\widehat{y})|Abs(\widehat{x}) + |\frac{\partial f}{\partial y}(\widehat{x},\widehat{y})|Abs(\widehat{y})$$

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:

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

computed $\widehat{z} = \dfrac{\widehat{x}\widehat{y}}{\widehat{x}+\widehat{y}}$

$$z = f(x,y) \quad \Rightarrow \frac{\partial f}{\partial x} = \frac{y^2}{(x+y)^2} \quad \frac{\partial f}{\partial y} = \frac{x^2}{(x+y)^2}$$
$$fx(\widehat{x},\widehat{y}) \approx 0.288103 \quad fy(\widehat{x},\widehat{y}) \approx 0.214599$$
$$\therefore Abs(f(\widehat{x},\widehat{y})) \approx 0.288103 \cdot Abs(\widehat{x}) + 0.214599 \cdot Abs(\widehat{y}) \approx 0.00251351$$

□

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 ≡ Cond

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

where we're assuming that $\widehat{x} \approx x$.

Therefore,

Cond $\to 1$ ill-conditioned
Cond $\ll 1$ well-conditioned.

□ In practical terms, however, it is not unusual to find that a condition number in the order of a decade is not at all bad. In systems of equations, in particular, the condition number can grow quite a bit and be overly pessimistic.

Example Evaluating a function: consider propagated data error when f is evaluated at $\widehat{x} = x + h$ instead of at x:

$$\begin{aligned}
\text{Abs error} &= f(x+h) - f(x) \approx hf'(x) \\
\text{Rel error} &= \frac{f(x+h) - f(x)}{f(x)} = h\frac{f'(x)}{f(x)} \\
\text{Cond } (x) &= \left|\frac{hf'(x)/f(x)}{h/x}\right| = \left|\frac{xf'(x)}{f(x)}\right|
\end{aligned}$$

Suppose $f(x) = e^x$

$$\begin{aligned}
\text{Absolute error } \equiv e_a &= e^{x+h} - e^x \approx he^x \\
\text{Relative error } \equiv e_r &= \frac{e^{x+h} - e^x}{e^x} \approx h\frac{e^x}{e^x} = h \\
\text{Cond}(x) &= |x|
\end{aligned}$$

□

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 $x \approx \pi/2$
$\widehat{x} = x + h$

$e_a = \cos(x+h) - \cos(x) \simeq -h\sin(x) \approx -h$

$e_r = -h\tan(x) \approx \infty$

$\therefore$ small changes in x near $\dfrac{\pi}{2}$ cause large relative changes in $\cos x$ *regardless of the method used in the computation.*

$$\begin{aligned} \cos(1.57079) &= 0.63267949 \cdot 10^{-5} \\ \cos(1.57078) &= 1.63267949 \cdot 10^{-5} \end{aligned}$$

$\therefore$ relative change in the output

$$\begin{aligned} \text{ratio of inputs} &= 1.00000636 \\ \text{ratio of outputs} &= 2.5805791 \end{aligned}$$

□

OK, so now we're ready to define a condition number more precisely. Again, this is a definition, not a mathematical notion, and hence, it has a certain amount of inherent idyosyncracies...namely, for large scale systems, it is overly pessimistic.

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 δx is also an m dimensional vector of small size, is the perturbation in y, call it δy which is of size n, is δy to within a multiplicative constant the same as δx? 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 ξ is some value between x and $x + \delta x$. We then form

$$\frac{\delta y}{y} \approx \frac{x f'(x)}{f(x)} \frac{\delta x}{x}.$$

The approximation happened in where we evaluate the derivative...we are assuming that δx is small but finite, by the way. The *relative error* is just the absolute value of this:

$$||\frac{\delta y}{y}|| \approx Cond(f;x)|\frac{\delta x}{x}|.$$

where the *CONDITION NUMBER* is

$$Cond(f;x) = |\frac{xf'(x)}{f(x)}|.$$

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.

□

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

$$\begin{aligned} \text{Cond}(f;x) &= \left|\frac{xf'(x)}{f(x)}\right| \\ &= \left|\frac{x \cdot \dfrac{-1}{x^2}}{\dfrac{1}{x}}\right| \\ &= 1 \end{aligned}$$

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 $\dfrac{\delta x}{x}$ which could be large if x is sufficiently small.

Now let's consider the function

$$y = g(x) = \sin(x)$$

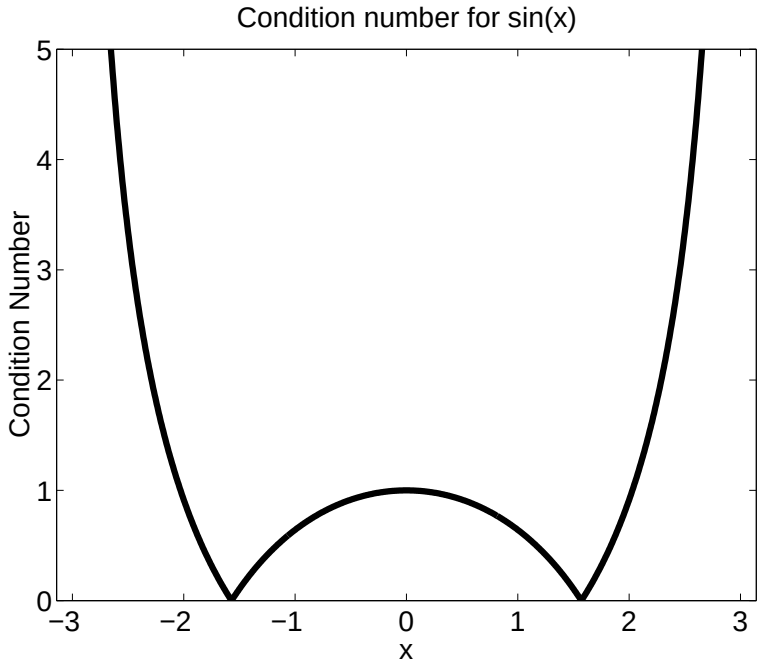

Figure 10: Condition number plotted against x for $\sin(x)$

which has the derivative

$$g'(x) = \cos(x).$$

The condition number is

$$\text{Cond}(g, x) = \left| \frac{x \cos(x)}{\sin(x)} \right| = \left| \frac{x}{\tan(x)} \right|. \tag{1}$$

In this case the condition number is dependent on the value of x. Figure 10 shows the condition number plotted against x. For small values of x (e.g. $x \in [\frac{-3\pi}{4}, \frac{3\pi}{4}]$) 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 lets 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

$$\begin{aligned} \text{Cond}(h; x) &= \left| \frac{xh'(x)}{h(x)} \right| \\ &= \left| \frac{x(4x+1)}{2x^2 + x - 1} \right| \\ &= \left| \frac{4x^2 + x}{2x^2 + x - 1} \right|. \end{aligned}$$

Again, this condition number is dependent on x. Figure 10 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

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

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.

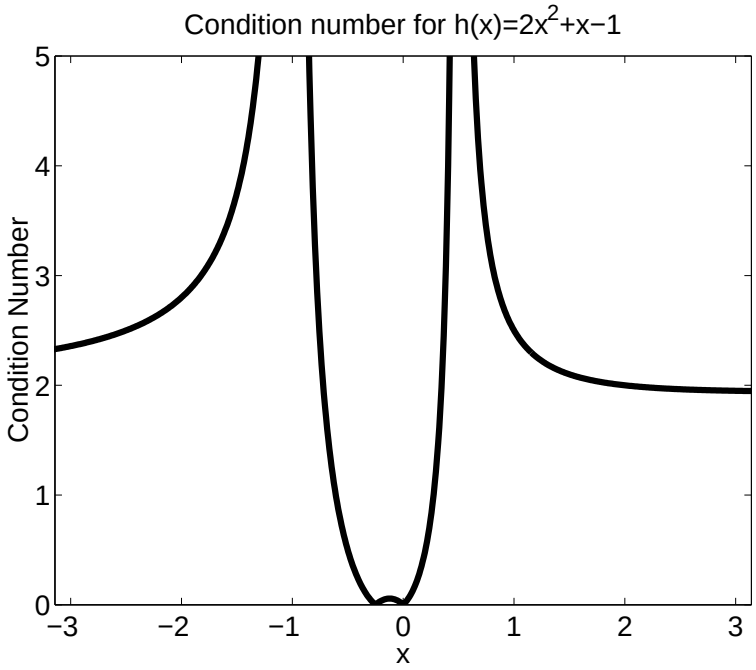

Figure 11: Condition number plotted against x for $h(x) = 2x^2 + x - 1$

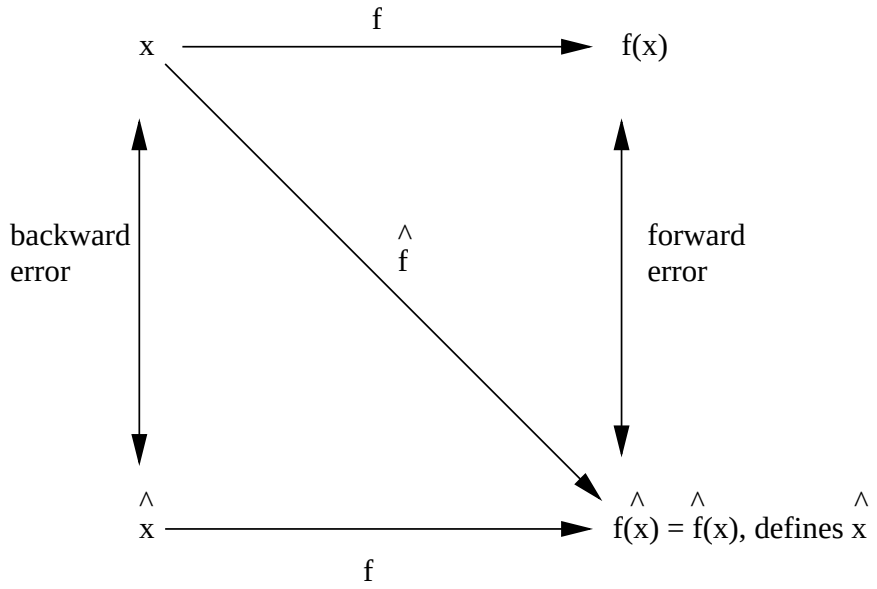

Figure 12: Schematic for the forward and backward error analysis.

4.1 Backward Error Analysis

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 12. 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 $x = 1$.

$$f(x) = e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 \cdots$$
$$\widehat{f}(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3$$

$$\boxed{\text{Forward error} = \widehat{f}(x) - f(x)}$$

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

$$f(\widehat{x}) = e^{\widehat{x}} \quad \therefore \quad \widehat{x} = \log(f(\widehat{x})) = \log(\widehat{f}(x))$$

$$f(x = 1) = 2.718282 \quad \widehat{f}(x) = 2.666667, \text{ to seventh decimal place.}$$
$$\widehat{x} = \log(2.666667) = 0.980829$$

The following are different and cannot be compared:

Forward error $\widehat{f}(x) - f(x) = -0.051615$ (OK)
Backward error $\widehat{x} - x = -0.019171$
(Ok, because output is correct for a slightly perturbed input.)

□

An algorithm is <u>stable</u> 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.

□

5 SOLUTION OF NONLINEAR EQUATIONS

<u>Methods</u>

1. Two Point Methods

2. One Point Methods

We'll also discuss Polynomial Root Finding later.

BASIC PROBLEM:

Find x such that $\quad f(x) = 0 \quad f : R \to R \cap C^1 \quad x \in R.$

In most cases it is not possible to solve this problem analytically. Most methods are iterative and require some initial guess(es). The most common are:

Two-point Methods: require 2 guesses
One-point Methods: require 1 guess

Two-point Methods:

Consider bracketing methods. These are inspired by the above two theorems.

General Technique in bracketing: if $f \in C[a, b]$ and $f(a)f(b) \leq 0$ then by the intermediate value theorem $\exists$ at least one 0 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 be bracket hopefully containing the root you want $f(\alpha_1) = 0$. Search and iterate making the brackets smaller until you hit the root.

5.1 Bisection Method:

A bracketing technique. see Figure 13

Bisection Algorithm

1. is $f(a)f(b) < 0$?
   $c = \frac{1}{2}(a + b)$

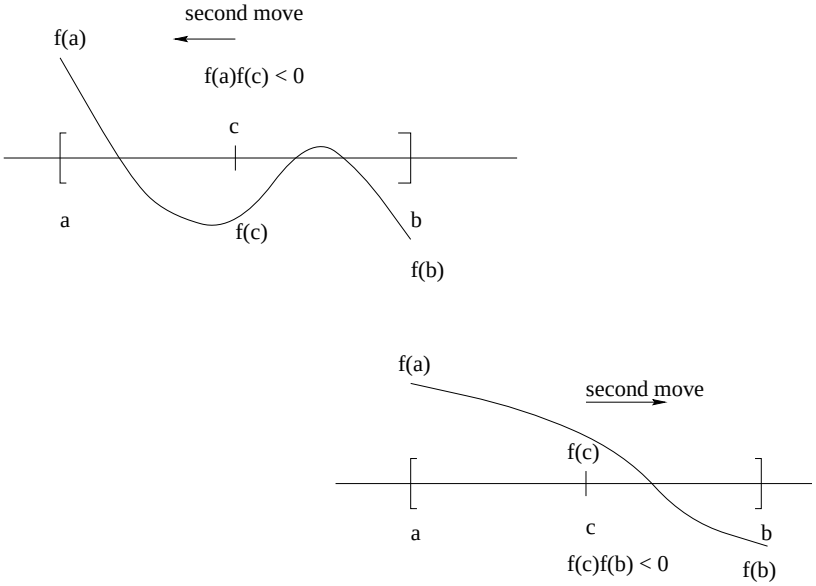

Figure 13: Second move directions according to the bracketing rule for two different cases.

2. is $f(a)f(c) < 0$?: % else $f(c)f(b) < 0$

   repeat

$$\begin{cases} \text{if } (f(a)f(c)) < 0) \\ \quad b = c \\ \text{else } a = c \\ \text{else } f(c) = 0 \end{cases}$$

   Remark: Unlikely to find $f(c) \equiv 0$. Due to roundoff ... so we use $|f(c)| < |\delta|, \delta \ll 1$.

3. Potential Round-off Problem

$$c_0 = \frac{a+b}{2}$$

$$c_1 = \frac{c_0 + b}{2} = \frac{1}{4}(a+3b)$$

$$c_2 = \frac{1}{2.4}(a+7b)$$

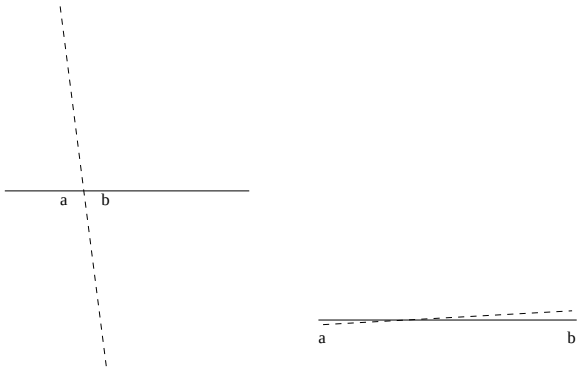

Figure 14: In the first figure the slope is large that you might get overflows, in the second, that you might get underflows.

$$c_3 = \frac{1}{2 \cdot 4 \cdot 8}(a + 15b)$$

One can see that the first term can become so small compared to the second term which can then lead to errors, due to roundoff. In general, always best to add small correction to a previous approximation. Could otherwise lead to $c \notin [a, b]$.

Implementation: use

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

instead of

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

4. 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 14

5. Implementation: use threee stopping criteria:

   (1) maximum number of bisections (avoids runaway jobs)

   (2) location error: $|b - a| < \epsilon$

   (3) accuracy of root: $|f(c)| < \delta$

   make code satisfy all three criteria.

   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

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

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 $0 \geq f(\lim_{n\to\infty} a_n) f(\lim_{a\to\infty} b_n) = f(r)f(r)$. 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: $c_n = \dfrac{a_n + b_n}{2}$, then

$$|r - c_n| \leq \frac{1}{2}(b_n - a_n) = 2^{-(n+1)}(b_0 - a_0)$$

□

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 $\lim\limits_{n\to\infty} a_n$ and $\lim\limits_{n\to\infty} b_n$ 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 $r \geq 50 \quad \therefore \quad \frac{|r - c_n|}{x_0} \leq 10^{-12}$

By 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)

It seems that the simplest methods often converge the slowest. This certainly seems to be the case here.

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

$$\boxed{\text{Assume } f(x) \text{ continuous such that } f(a)f(b) < 0}$$

$\frac{y - f(b)}{x - b} = \frac{f(a) - f(b)}{a - b}$This gives the formula for the secant line

pick the intercept $y = 0$, then the next approximation is

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

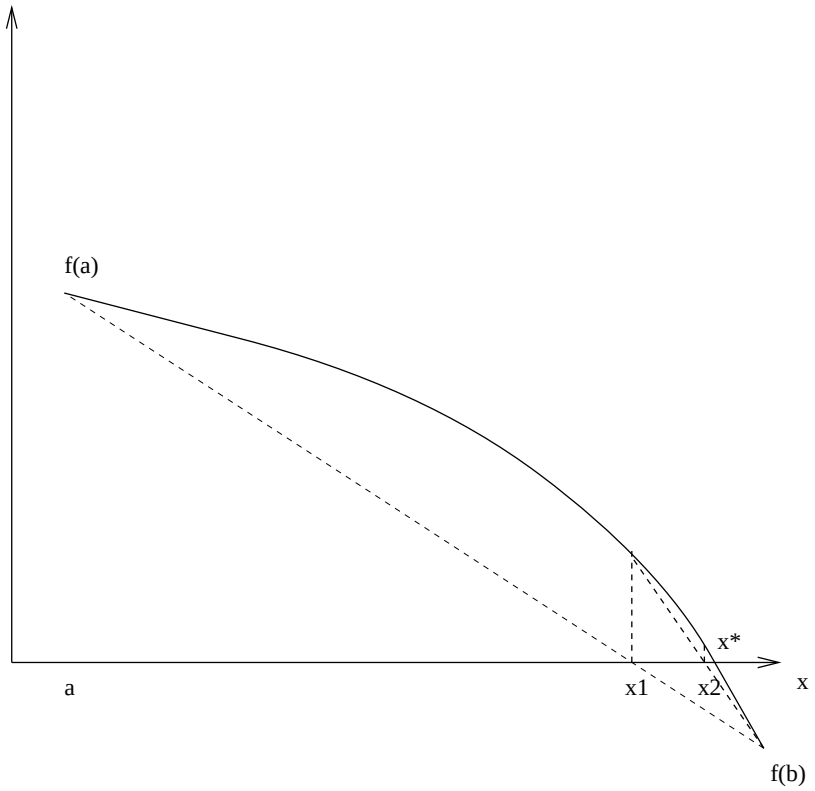

Figure 15: Regula-Falsi Algorithm.

The algorithm is depicted in Figure 15.

x_1 is the first approximation to x^*

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 $x^\star \in [x_1, b]$. Suppose $f(x_1)f(b) < 0$, then

$$x_2 = \frac{x_1 f(b) - b f(x_1)}{f(b) - f(x_1)}, \text{ etc.}$$

This is the pseudo code for the Regula Falsi method:

```
User inputs: a,b, max, epsilon
Initialize: iteration = 0
              xold = b       //to start out

Inside a  loop with condition that iteration <=max:

  while(iteration <=max)
compute f(a)
compute f(b)

x = a - ((f(a)*(b-a)) / (f(b)-f(a))
            increment iteration

Test:
if |x-xold| < epsilon*|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
                end
            end
      end

stop
```

A potential problem: what should be the stopping criteria (Recall that in bisection we had intervals getting smaller)?

We could use

$$|x_n - x_{n-1}| < \varepsilon \tag{2}$$

and

$$|f(x_n)| < \delta. \tag{3}$$

however these can lead to some problems:

Suppose $|x_n - x_{n-1}| = \dfrac{1}{n}$, which is satisfied by (2) when $n \geq \dfrac{1}{\varepsilon}$. The problem will be that $\{x_n\}$ will actually diverge.

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

$$\text{whereas } |x^* - x_n| < 10^{-2} \quad \text{only for} \quad n > 100$$

So the test will signal convergence prematurely!

Instead the stopping criteria should be

$$\boxed{|x_n - x_{n-1}| < \varepsilon |x_n|}$$

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

5.3 Newton Raphson Method (N-R Method)

Problem Statement': Find $x = p$ such that $f(x) = 0$

Suppose $f \in C^2[a,b]$. Let $\overline{x} \in [a,b]$ be an approximation to p such that $f'(\overline{x}) \neq 0$ and $|\overline{x} - p|$ is"small." Then

$$f(x) = f(\overline{x}) + (x - \overline{x})f'(\overline{x}) + \frac{(x-\overline{x})^2}{2}f''(\xi(x)) + O(|x-\overline{x}|^3) \tag{4}$$

where $\xi(x)$ lies between x and $\overline{x}$. Since $f(p) = 0$, with $x = p$, (4) gives

$$0 = f(\overline{x}) + (p - \overline{x})f'(x) + \frac{(p-\overline{x})^2}{2}f''(\xi(p)). \tag{5}$$

If $|p - \overline{x}|$ is small, then $|p - \overline{x}|^2$ smaller, and

$$0 \approx f(\overline{x}) + (p - \overline{x})f'(\overline{x})$$

is a good approximation. Solving for p:

$$p \approx \overline{x} - \frac{f(\overline{x})}{f'(\overline{x})}$$

The N-R method begins with an estimate p_0 and generates a sequence $\{p_n\}$,

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

See Figure 16 for the algorithm.

Newton-Raphson Algorithm

inputs initial guess p_0, tolerance TOL, maximum iterations

1. Set $i = 1$

2. While $i \leq N_0$ do Steps 3 - 6

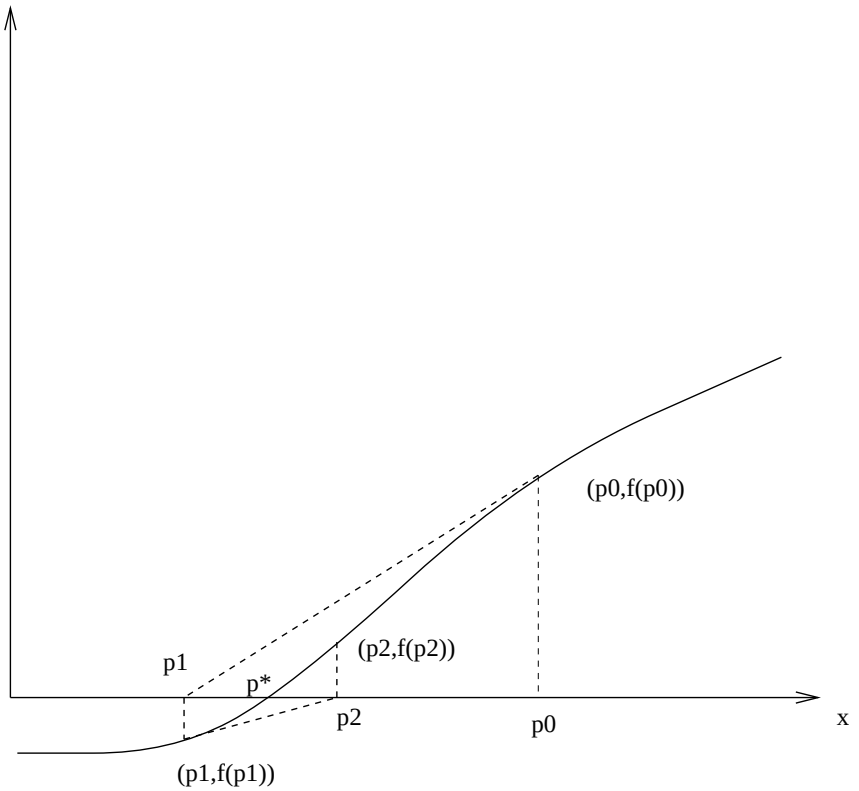

Figure 16: Newton Raphson Algorithm

3. Set $p = p_0 - f(p_0)/f'(p_0)$ % to compute p_i

4. If $|p - p_0| <$ TOL then
   OUTPUT p
   STOP

5. Set $i = i + 1$

6. Set $p_0 = p$ % update p_0

7. Output ('Method failed after N_0 iterates')
   STOP

Pros: Faster than 2-point methods: the bisection and regula-falsi method are linear, secant method (see 5.5) is superlinear. N-R Method has quadratic convergence.

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

b) Not guaranteed to always converge.

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 and f' continuous. Then

$$\begin{aligned} e_{n+1} &= p_{n+1} - p = p_n - \frac{f(p_n)}{f'(p_n)} - p \\ &= e_n - \frac{f(p_n)}{f'(p_n)} = \frac{e_n f'(p_n) - f(p_n)}{f'(p_n)} \end{aligned} \tag{6}$$

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 ξ_n is between p_n and p. Rearranging terms we see that

$$e_n f'(p_n) - f(p_n) = \frac{1}{2} f''(\xi_n) e_n^2 \tag{7}$$

Which we substitute into 6) to get

$$e_{n+1} = \frac{1}{2}\frac{f''(\xi_n)}{f'(p_n)} e_n^2 \approx \frac{1}{2}\frac{f''(p)}{f'(p)} e_n^2 = C e_n^2 \tag{8}$$

Therefore, the algorithm has at least quadratic convergence rate.

Note that we haven't established convergence itself. Just the rate. By (7), 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 $c(\delta) = \frac{1}{2} \max_{|x-p|\leq\delta} |f''(x)| \Big/ \min_{|x-p|\leq\delta} |f'(x)|, \quad \delta > 0$

Select δ small enough so that the denominator is positive, and then if necessary, decrease δ so that $\delta c(\delta) < 1$. This is possible because as $\delta \to 0$, then $c(\delta)$ converges to $\frac{1}{2}|f''(p)| \Big/ |f'(p)|$, and $\delta c(\delta)$ converges to 0.

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

Suppose we start the N-R Method with p_0 satisfying $|p_0 - p| \leq \delta$. Then $|e_0| \leq \delta$ and $|\xi_0 - p| \leq \delta$ then

$$\frac{1}{2}|f''(\xi_0)| \Big/ |f'(p_0)| \leq c(\delta)$$

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

Taking (7) $|p_1 - p| = |e_1| \leq e_0^2 c(\delta) = |e_0||e_0|c(\delta) \leq |e_0|\delta c(\delta) = |e_0|\rho < |e_0| \leq \delta$

$\therefore p_1$ lies within δ distance to p. Repeating,

$$\begin{aligned} |e_1| &\leq \rho |e_0| \\ |e_2| \leq \rho |e_1| &\leq \rho^2 |e_0| \\ |e_3| &\leq \rho^3 |e_0| \\ &\vdots \\ |e_n| &\leq \rho^n |e_0| \end{aligned}$$

Since $0 \leq \rho < 1 \quad \lim_{n\to\infty} \rho^n = 0 \therefore \lim_{n\to\infty} e_n = 0$

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

$$|p_{n+1} - p| \leq C(p_n - p)^2 \quad n \geq 0$$

□

In some situations Newton Method is guaranteed to converge from any arbitrary starting point:

Theorem 2: 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 (8). □

Example Efficient Method of computing square root of number:

Let $R > 0 \quad$ 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 N-R Method. Consider

$$\begin{aligned} p_{n+1} &= p_n - f(p_n) \Big/ g(p_n) \\ \text{where } g(x) &= \frac{[f(x + f(x)) - f(x)]}{f(x)} \\ \text{This requires } g &\in C^3[a,b] \\ &\quad \text{for quadratic convergence} \end{aligned}$$

So the value of this algoritm primarily lies in the fact that no $f'(x)$ is needed.

□

5.5 Secant Method

Like the Newton method with a secant approximation to $f'(x)$. Newton's Method requires $f'(x)$

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

since $f'(x) = \lim\limits_{n \to x} \dfrac{f(h) - f(x)}{h - x}$

The tangent is replaced by secant:

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

So we need two starting values in this method.

Rate of Convergence:

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

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

□

The algorithm for the Secant method is depicted in Figure 17.

Secant Method Algorithm

Find $f(x) = 0$ given p_0, p_1

input p_0, p_1 tolerance, N_0.
output p, or failure message

1. Set $i = 2$
   $q_0 = f(p_0)$

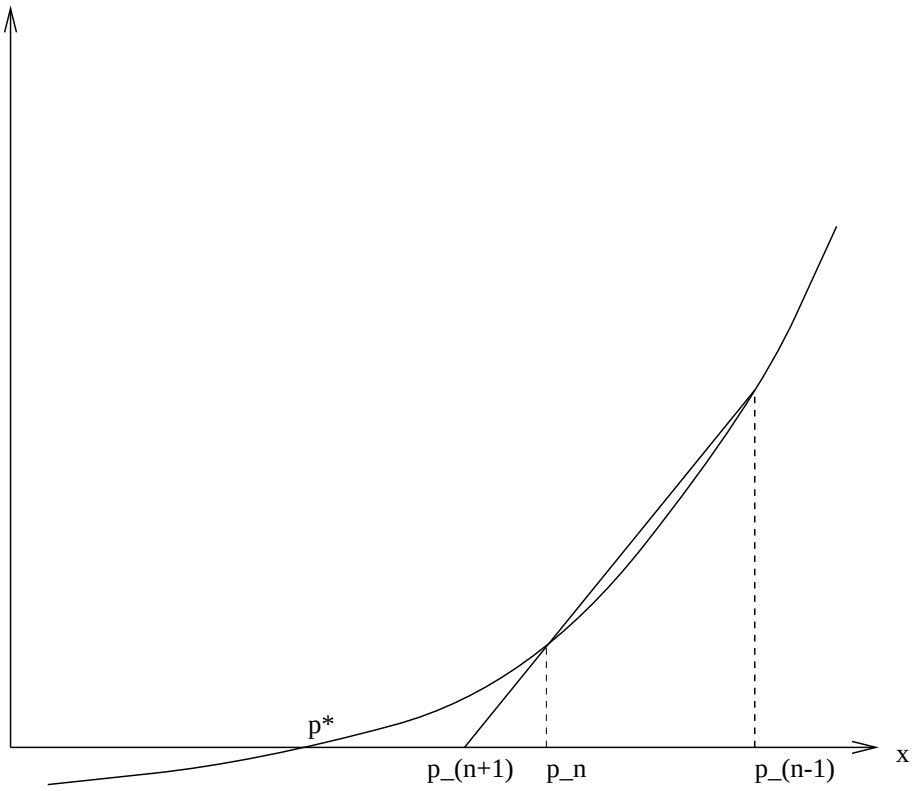

Figure 17: Secant Method Algorithm

$q_1 = f(p_1)$

2. While $i \leq N_0$ do 3 - 6:

3. Set $p = p_1 - q_1(p_1 - p_0)\Big/(q_1 - q_0)$ % compute p_i

4. if $|p - p_1| <$ TOL then
   output p % done
   Stop

5. $i = i + 1$

6. Set $p_0 = p_1$ % update p_0, q_0, p_1, q_1
   $q_0 = q_1$
   $p_1 = p$
   $q_1 = f(p)$

7. Output ('Method failed after N_0 iterates).
   STOP

□

5.6 Fixed Point Iteration

def: 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 $g(x) = x - \sin \pi x$ has 2 fixed points in $[0,1]$ $\quad x = 0$, $x = 1$. Plot this to see it.

□

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

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

then $|p - q| = |g(p) - g(q)| = |g'(\xi)|p - q|| \leq k|p - q| < |p - q|$ which is a contradiction.

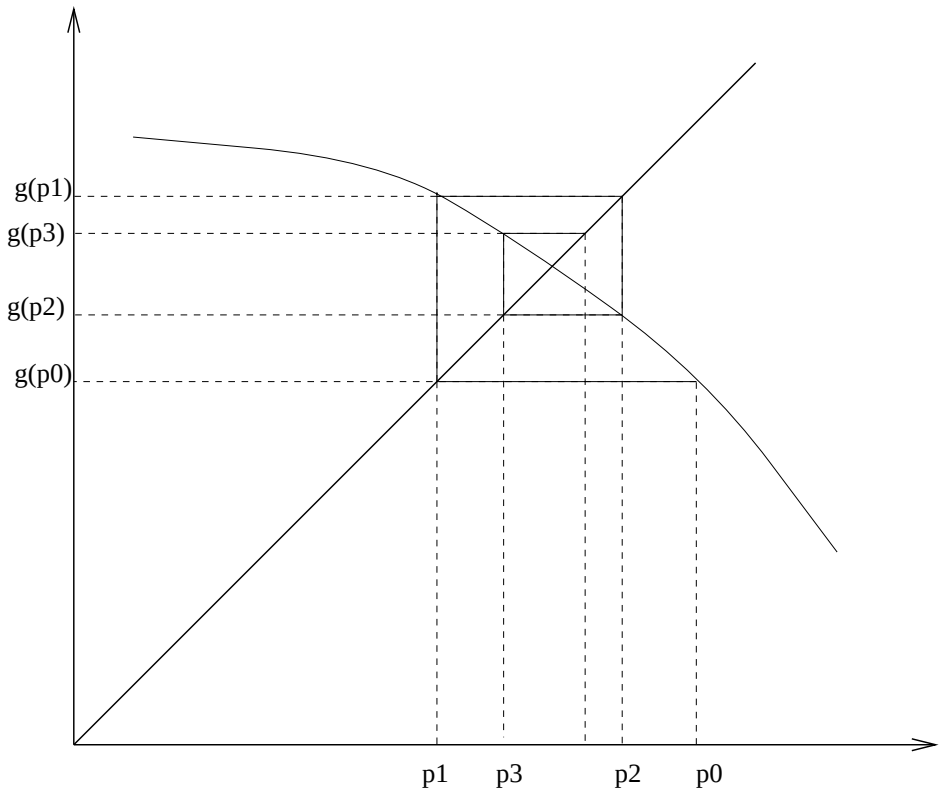

Figure 18: Fixed point iteration illustrated graphically

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 $g(p) = p$. The questions 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 g is continuous then by the theorem above:

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

The algorithm is depicted in Figure 18

Fixed Point Algorithm

Input: p_0, TOL, N_0 (max number of iterations)
Output: p or message of failure

1. Set $i = 1$

2. While $i \leq N_0$

3. Set $p = g(p_0)$ % Compute p_i

4. if $|p - p_0| <$ TOL then
   output (p); % found
   Stop

5. Set $i = i + 1$

6. Set $p_0 = p$ Update p.

7. Output (Iterations exceeded. $i > N_0$)
   END

□

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 ξ_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

$$\lim_{n\to\infty} \frac{p_{n+1} - p}{p_n - p} = \lim_{n\to\infty} g'(\xi_n) = g'(p) \text{ and}$$

$$\lim_{n\to\infty} \frac{|p_{n+1} - p|}{|p_n - p|} = |g'(p)|$$

therefore, the fixed point iteration converges linearly if $g'(p) \neq 0$.

□

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:

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

□

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

$$|g'(x)| \leq k < 1 \quad \forall x \in (a, b). \tag{9}$$

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 (9) 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

$$|p_n - p| \leq k|p_{n-1} - p| \leq k^2|p_{n-2} - p| \leq \cdots k^n|p_0 - p|. \tag{10}$$

Since $k < 1$

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

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

$$|p - p_n| \leq k^n \max\{p_0 - a, b - p_0\} \quad \text{(a)}$$
and
$$|p - p_n| \leq \frac{k^n}{1 - k}|p_1 - p_0| \qquad \forall n$$
$$\geq 1 \quad \text{(b)}$$

Proof: (a) follows from (10):

$$|p_n - p| \leq k^n|p_0 - p| \leq k^n \max\{p_0 - a, b - p_0\} \text{ since } p \in [a, b].$$

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 $m > n \geq 1$,

$$\begin{aligned} |p_m - p_n| &= |p_m - p_{m-1} + p_{m-1} - p_{m-2} + p_{m-2} - \cdots + p_ \\ &\leq |p_m - p_{m-1}| + |p_{m-1} - p_{m-2}| + \cdots |p_{n+1} - p_r \end{aligned}$$

$$\begin{aligned} &\leq k^{m-1}|p_1 - p_0| + k^{m-2}|p_1 - p_0| + \cdots k^n|p_1 - p_0| \\ &= k^n(1 + k + k^2 \ldots k^{m-n-1})|p_1 - p_0|. \end{aligned}$$

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 $\sum_{i=0}^{\infty} k^i = \frac{1}{1-k}$ then

$$|p - p_n| \leq \frac{k^n}{1-k}|p_1 - p_0|.$$

□

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 19. Get matlab code used in the example.

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 $g(x)$

$$\begin{aligned} g'(x) &= \frac{1}{3}(3x + 18)^{\frac{-2}{3}} \times 3 \\ &= \frac{1}{(3x + 18)^{\frac{2}{3}}} \end{aligned}$$

$$|g'(x)| = \frac{1}{(3x + 18)^{\frac{2}{3}}} \leq \frac{1}{18^{\frac{2}{3}}} < 1 \text{ for } x \in [0, \infty)$$

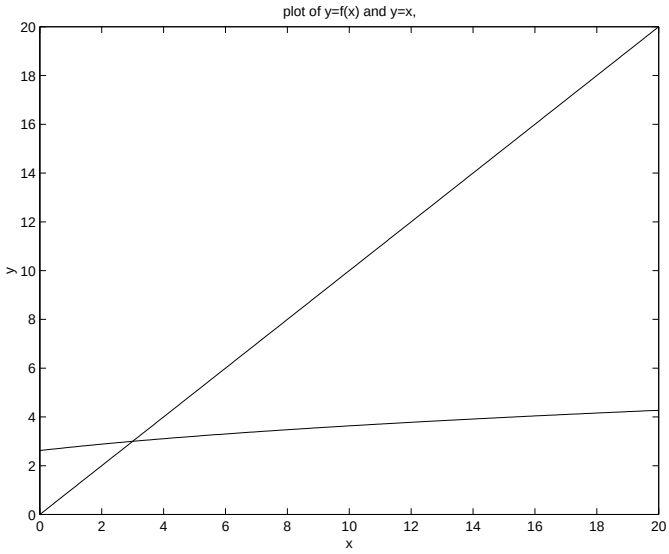

Figure 19: Plot of $g(x)$

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 20a and Figure 20b illustrate the iteration history and the logarithm of the error, for the case starting with $x_0 = 0$. Figure 21a and Figure 21b illustrate the iteration history and the logarithm of the error, for a case starting with $x_0 = 100$. Finally, Figure 22a and Figure 22b illustrate the iteration history and the logarithm of the error, for a case starting with $x_0 = 10000$.

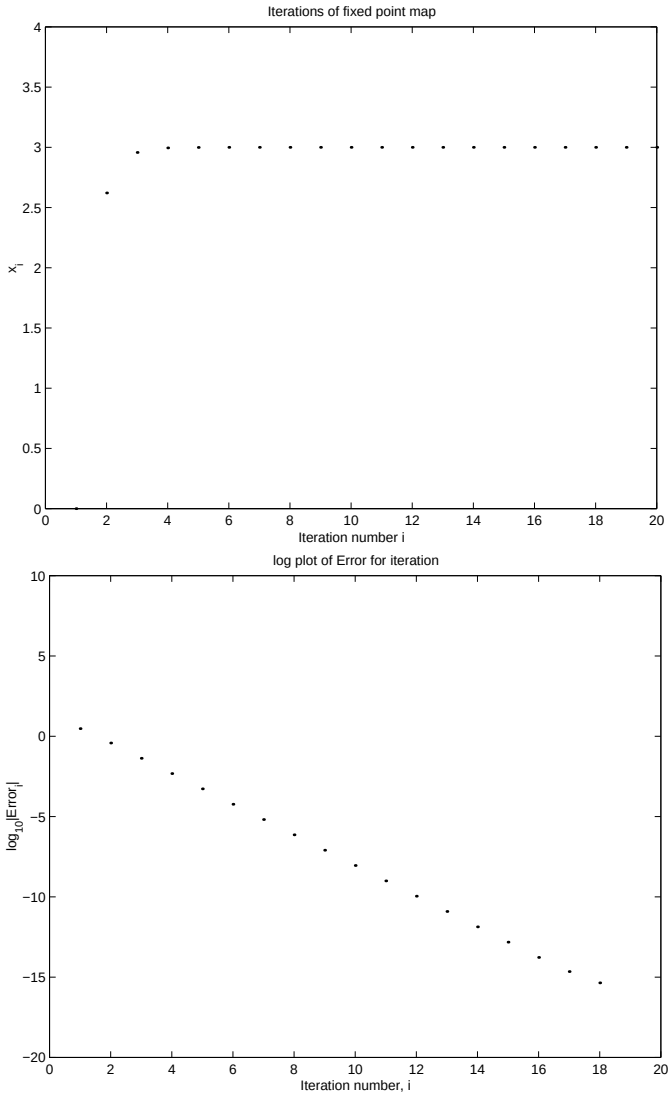

Figure 20: Iteration history and the logarithm of the error. Case starting with $x_0 = 0$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$

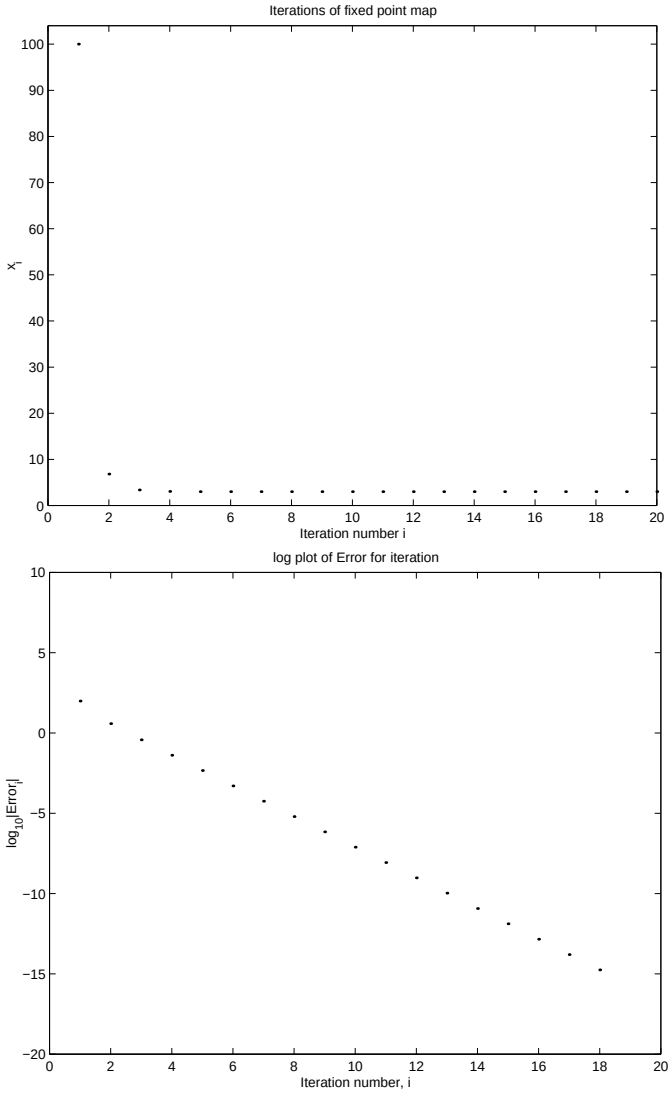

Figure 21: Iteration history and the logarithm of the error. Case starting with $x_0 = 100$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$

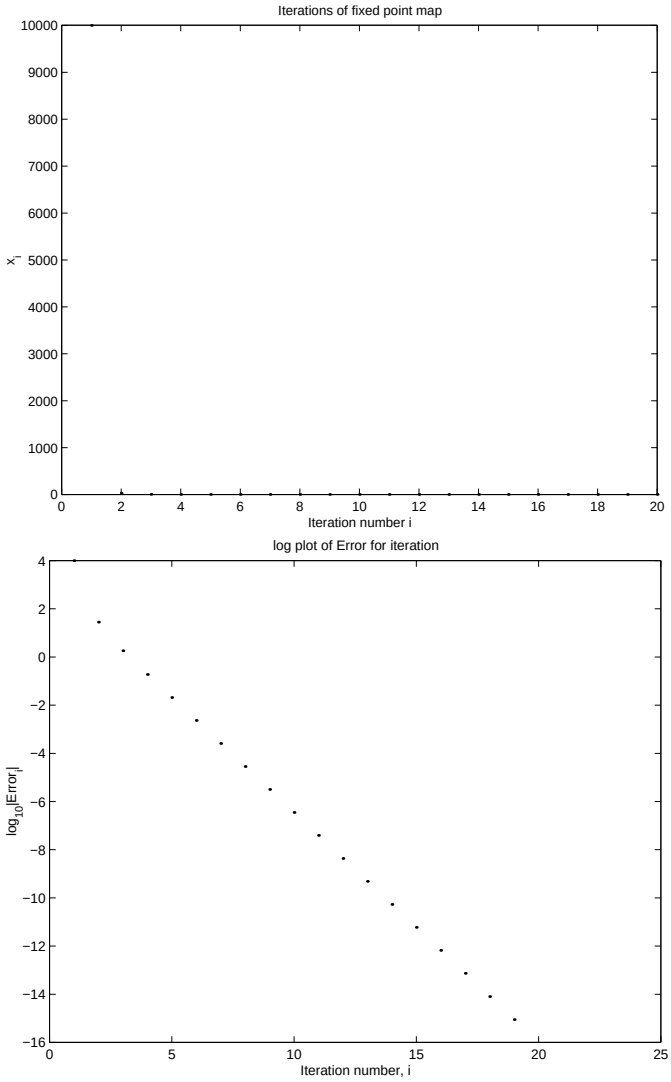

Figure 22: Iteration history and the logarithm of the error. Case starting with $x_0 = 10000$. $g(x) = (3x + 18)^{\frac{1}{3}}$ for $x \in [0, \infty)$

x_0	0	100	10000
x_1	2.6207	6.8256	31.0785
x_2	2.9573	3.3760	4.8093
x_3	2.9952	3.0412	3.1889
x_4	2.9995	3.0046	3.0208
x_5	2.9999	3.0005	3.0023
x_6	3.0000	3.0001	3.0002
x_7	3.0000	3.0000	3.0000
x_8	3.0000	3.0000	3.0000
x_9	3.0000	3.0000	3.0000
x_{10}	3.0000	3.0000	3.0000

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, $x = 3$.

□

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 α. Get matlab code used in the example.

Now lets 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 α 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 α 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 α 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 α less than two. What is happening in these cases?

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 tell 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 α 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 $g(x) = x$.

$$\begin{aligned} x &= \frac{x^2 - 2x + 1}{\alpha} \\ \alpha x &= x^2 - 2x + 1 \\ 0 &= x^2 - (2 + \alpha)x + 1 \end{aligned}$$

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. □

5.6.1 Contraction Mapping

Let $C \subseteq R$ and $F : C \to C$ F is contractive if $\exists$ $\lambda < 1$ such that

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

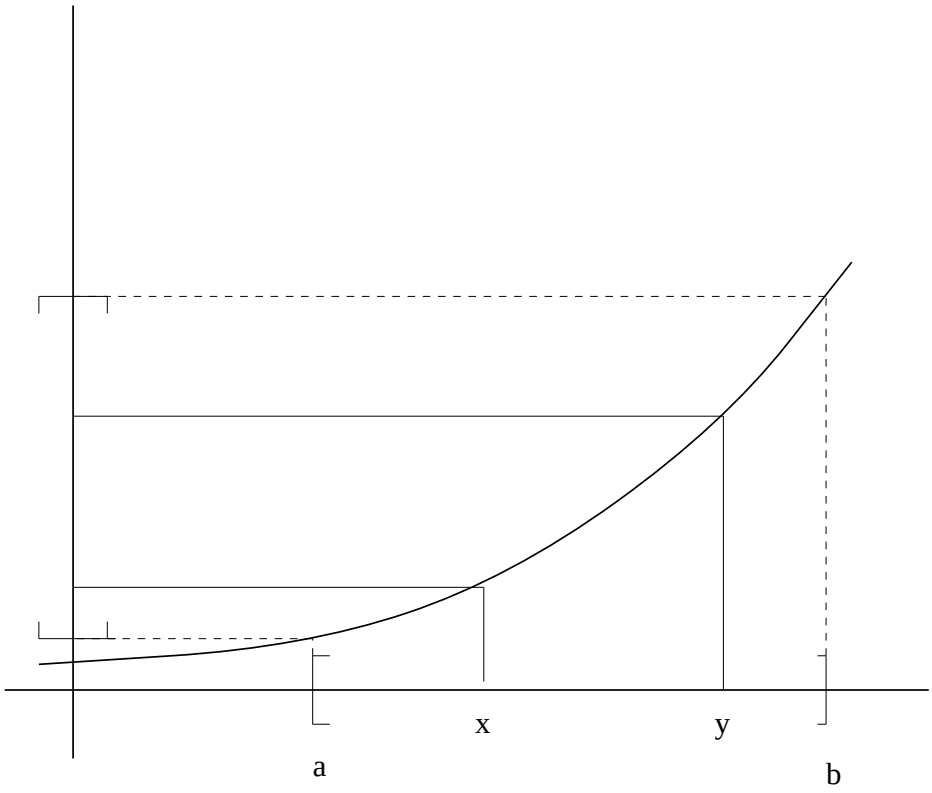

Figure 23: Contraction Mapping. The range of $f(x)$ between $x = a$ and $x = b$ is smaller in length than $b - a$.

See Figure 23 for a simple contraction example

Contractive Mapping Theorem: Let $C \subseteq R$ 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 $x_0 \in C$.

□

Remark In homework you will need to prove that when $F : [a, b] \to \mathbf{R}^1$, if F' is continuous and $|F'(x)| < 1$ on $[a, b]$ then F is a contraction map.

$$F'(x_n) = \lim_{x_{n+1} \to x_n} \frac{F(x_{n+1}) - F(x_n)}{x_{n+1} - x_n}$$

$$\text{if } |F'(x)| < 1 \text{ for } x \in [a, b] \text{ then}$$

$$\lim_{x_{n+1} \to x_n} |F(x_{n+1}) - F(x_n)| < \lim_{x_{n+1} \to x_n} |x_{n+1} - x_n|$$

considering this, can iterative map:

$$|F(x_{n+1}) - F(x_n)| < |x_{n+1} - x_n| \quad \therefore \text{contractive.}$$

F has a fixed point if $F : [a, b] \to [a, b]$ and it will be unique if $|F'(x)| < 1$ for all $x \in (a, b)$; so either $[a, b]$ is the whole real line, or there is no guarantee of a fixed point.

□

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 $|g'(x)| \leq k < 1 \qquad \forall x \in I$. 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

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

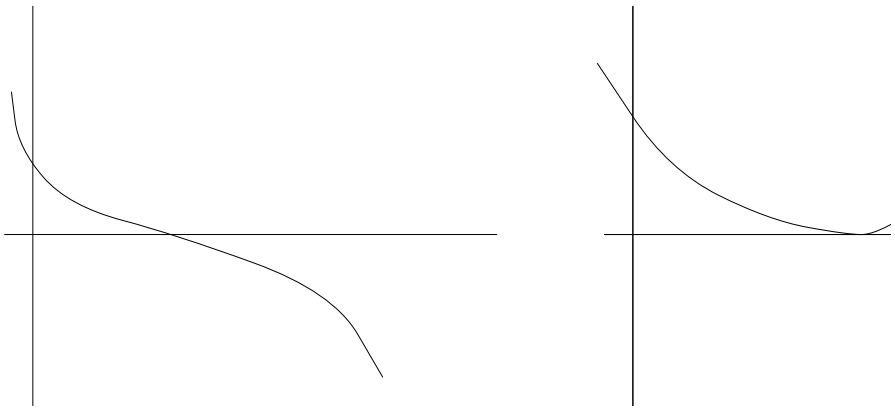

Figure 24: (a) Simple Root, (b) Multiple Root.

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.

□

Remark 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 24 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 $(a,b) \iff f(p) = 0$ but $f'(p) \neq 0$.

2) $f \in C^m[a,b]$ has a root of m multiplicity $p \iff$

$$0 = f(p) = f'(p) = f''(p) \cdots f^{(m-1)}(p) \text{ but } f^m(p) \neq 0.$$

□

Method for handling multiple zeros:

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

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

$$\mu(x) = \frac{(x-p)^m q(x)}{m(x-p)^{m-1}q(x) + (x-p)^m q'(x)} = \frac{(x-p)q(x)}{\underbrace{mq(x) + (x-p)q'(x)}_{\text{has root } p \text{ but multiplicity } 1}}$$

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! COULD BE A PROBLEM. So although we have a way to do the computation, we still have to be careful.

□

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

5.7 Zeros of Polynomials

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

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

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.

□

Corollary: If $P(x)$ is a polynomial of degree $n \geq 1$, then there are a unique set of constants $x_1, x_2, \cdots x_k$, (possibly complex), and a unique set of positive integers $m_1, m_2, \cdots m_k$, 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 $P = (x - x_1) * Q$ for some polynomial Q of degree $n - 1$. Q in turn has a root, which can be factored .... □

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:

5.7.1 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

$$Q(x) = b_n x^{n-1} + b_{n-1}x^{n-2} \cdots b_2 x + b_1$$
$$\text{then, } P(x) = (x - x_0)Q(x) + b_0$$

Proof: Direct Calculation

□

Example

$4x^4 + 13x^3 - x + 8 \quad$ at $x_0 = -3$
here $n = 4 \quad b_n = a_n = 4$
Remember $b_k = a_k + b_{k+1}x_0$

Using this formula we can calculate the coefficients b_i by induction:

$$\begin{aligned} b_3 &= a_3 + b_4 x_0 = 13 + 4(-3) = 1 \\ b_2 &= a_2 + b_3 x_0 = 0 + 1(-3) = -3 \\ b_1 &= a_1 + b_2 x_0 = -1 + -3(-3) = 8 \\ b_0 &= P(x_0) = a_0 + b_1 x_0 = 8 + 8(-3) = -16 \end{aligned}$$

This gives us the following polynomials:

$$\begin{aligned} P(x) &= (x - x_0)Q(x) + b_0 = (x+3)Q(x) - 16 \\ Q(x) &= b_n x^{n-1} + b_{n-1}x^{n-2} + \ldots + b_2 x + b_1 \\ &= b_4 x^3 + b_3 x^2 + b_2 x + b_1 = 4x^3 + x^2 - 3x + 8 \\ P(x) &= (x+3)(4x^3 + x^2 - 3x + 8) - 16 \end{aligned}$$

□

TABLE FORM:

Coefficient of $x_0 = -3$	x^4	x^3	x^2	x^1	x^0
	$a_4 = 4$	$a_3 = 13$	$a_2 = 0$	$a_1 = -1$	$a_0 = 8$
		$b_4x_0 = -12$	$b_3x_0 = -3$	$b_2x_0 = 9$	$b_1x_0 = -24$
	$b_4 = 4$	$b_3 = 1$	$b_2 = -3$	$b_1 = 8$	$b_0 = -16$

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

$$\text{Since } P(x) = (x - x_0)Q(x) + b_0$$
$$\text{where } Q(x) = b_n x^{n-1} + b_{n-1}x^{n-2} \cdots b_2 x + b_1$$

differentiating,

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

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

ALGORITHM:

To find $P(x_0)$ and $P'(x_0)$:

input:	degree n; coeff's	$a_0, a_1 \cdots a_n$; x_0
output:	$y = P(x_0)$; $z = P'(x_0)$	
Step 1	$y = a_n$	% compute $b_n$for P
	$z = a_n$	% compute b_{n-1} for Q
Step 2	for $j = n-1, n-2 \cdots 1$	
	$y = x_0 y + a_j$	% compute b_j for P
	$z = x_0 z + y$	%compute b_{j-1} for Q
Step 3	Set $y = x_0 y + a_0$	% compute b_0 for P
Step 4	output (y, z)	
	END.	

□

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 synthetic division. These are used to evaluate $P(x_n)$ and $P'(x_n)$ for each iterate x_n. With $x_0 = -3$ as initial guess, we obtained previously $P(-3)$ in previous example.

	a_4	a_3	a_2	a_1	a_0
$x_0 = -3$	4	13	0	-1	8
		$b_4x_0 := -12$	$b_3x_0 = -3$	$b_2x_0 = 9$	$b_1x_0 = -24$
	$b_4 = 4$	$b_3 = 1$	$b_2 = -3$	$b_1 = 8$	$b_0 = 16 = P(-3)$

$$Q(x) = 4x^3 + x^2 - 3x + 8 \text{ and } P'(-3) = Q(-3)$$

So $P'(-3)$ can be found by evaluating $Q(-3)$:

$x_0 = -3$	4	1	-3	8
		$b'_4x_0 = -12$	$b'_3x_0 = 33$	$b'_2x_0 = -90$
	$b'_4 = 4$	$b'_3 = -11$	$b'_2 = 30$	$b'_1 = -82 = Q(-3) = P'(-3)$

Then, by Newton Raphson,

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

Repeatingthis procedure yields x_2:

$x_1 = -3.1951$	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	$-124.7513 = Q(x_1) = P'(x_1)$	

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.

□

Note that Q depends on the approximation and will change from iterate to iterate.

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

$$P(x) = (x - x_n)Q(x) + b_0 = (x - x_n)Q(x) + P(x_n) \approx (x - x_n)Q(x)$$
$$\text{so } x - x_n \text{ is an approximate factor of } P.$$

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

$$P(x) \approx (x - \widehat{x}_1)Q_1(x)$$

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

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

The difficulty with deflation is that Newton-Raphson applied repetitively compounds errors. The larger k, the worse $\widehat{x}_{k+1}$, 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 $\widehat{x}_2, \widehat{x}_3, \cdots \widehat{x}_k$ 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 $\widehat{x}_i$'s to define a good search interval for each root.

Now, suppose the roots 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.

5.7.2 Müller's Method

Developed in (1956). Used to find zeros of arbitrary analytic functions (particularly useful in finding roots of polynomials.) It is a variation of the secant method (see 5.5). The process is illustrated in Figure 26. 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 26. 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 $(x_0, f(x_0 1)),\ (x_1, f(x_1)),\ (x_2, f(x_2))$

Determine a, b, c:

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

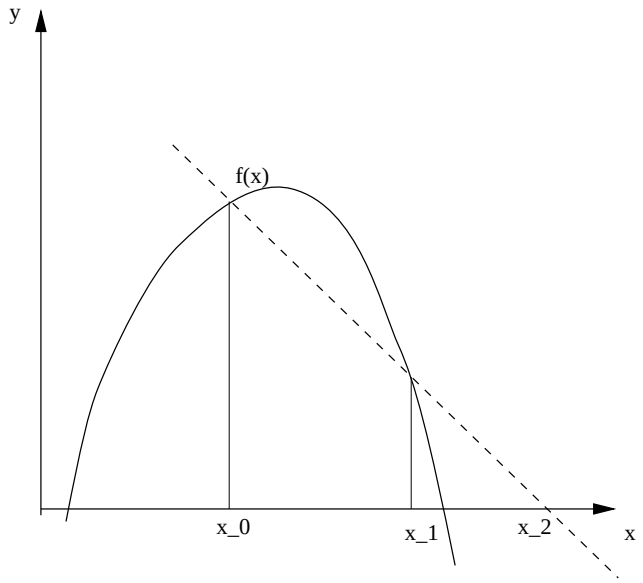

Figure 25: Finding zeros with the Secant Method. We look for the intersection of the secant and the x-axis. The approximate root is x_2.

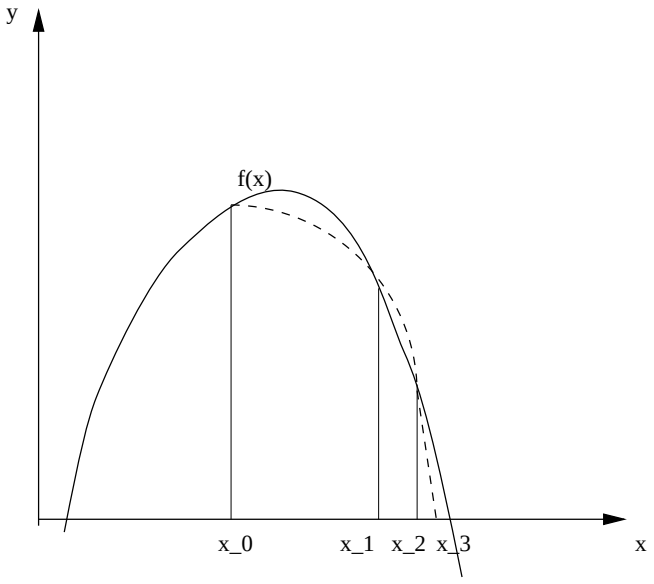

Figure 26: Finding zeros using Müller's method. We locally fit a parabola and look at the intersection of the parabola with the x-axis.

$$a = \frac{(x_1 - x_2)[f(x_0) - f(x_2)] - (x_0 - x_2)[f(x_1) - f(x_2)]}{(x_0 - x_2)(x_1 - x_2)(x_0 - x_1)}$$

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:

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

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

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

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

Algorithm: Müller find x such that $f(x) = 0$ with x_0, x_1, x_2

input: x_0, x_1, x_2, TOL, N_0
output: approximate x or failure

Step 1 Set $h_1 = x_1 - x_0$
$h_2 = x_2 - x_1$
$\delta_1 = (f(x_1) - f(x_0))/h_1$
$\delta_2 = (f(x_2) - f(x_1))/h_2$;
$d = (\delta_2 - \delta_1)/(h_2 + h_1)$
$i = 2$

Step 2 While $i \leq N_0$ *do* Steps 3 - 7

Step 3 $b = \delta_2 + h_2 d$
$D = \sqrt{b^2 - 4f(x_2)d}$ % may be complex

Step 4 if $|b - D| < |b + d|$
then set
$E = b + D$ else set $E = b - D$

Step 5 $h = -2f(x_2)/E$
$p = x_2 + h$

Step 6 if $|h| < TOL$ then
output p
STOP

Step 7 $x_0 = x_1$
$x_1 = x_2$
$x_2 = p$
$h_1 = x_i x_0$
$h_2 = x_2 - x_1$
$\delta_1 = (f(x_1) - f(x_0))/h_1$
$\delta_2 = (f(x_2) - f(x_1))/h_2$
$d = (\delta_2 - \delta_1)/(h_1 + h_2)$
$i = i + 1$
Step 8 output ('Failure')
END.

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 $P(x) = a(x+1)^2 + b(x+1) + c$

$$c = f(x_2) = 1 + 1/e \approx 1.3679$$

$$\begin{aligned} b &= \frac{(x_0-x_2)^2[f(x_1)-f(x_2)]-(x_1-x_2)^2[f(x_0)-f(x_2)]}{(x_0-x_2)(x_1-x_2)(x_0-x_1)} \\ &= \frac{4(1-1/e)-(e-1/e)}{2} \approx 0.0890 \end{aligned}$$

$$\begin{aligned} a &= \frac{(x_1-x_2)[f(x_0)-f(x_2)]-(x_0-x_2)[f(x_1)-f(x_2)]}{(x_0-x_2)(x_1-x_2)(x_0-x_1)} \\ &= \frac{(e-1/e)-2(1-1/e)}{2} \approx 0.5431 \end{aligned}$$

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

$$x_3 - x_2 = x_3 + 1 = \frac{-2c}{b+\sqrt{b^2-4ac}} \approx -0.0820 + 1.5849i$$

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 $x_4 - x_3 = -2c/(b \pm \sqrt{b^2-4ac})$. 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:

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

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:

n	x_n
0	1.0000
1	0.0000
2	-1.0000
3	-1.0820 + 1.5849i
4	-1.2735 + 2.8505i
5	-0.3814 + 3.7475i
6	0.1973 +3.1457i
7	-0.0168 + 3.1571i
8	0.0000 + 3.1421i
9	0.0000 + 3.1416i

6 Norms of Functions

$$
\begin{aligned}
L_1 - \text{norm} \qquad ||f-q||_1 &= \int_a^b |f(x)-q(x)|w(x)dx \\
L_2 - \text{norm} \qquad ||f-q||_2 &= \left\{ \int_a^b |f(x)-q(x)|^2 w(x)dx \right\}^{\frac{1}{2}} \\
L_\infty - \text{norm} \qquad ||f-q||_\infty &= \max_{a \le x \le b} |f(x)-q(x)|
\end{aligned}
$$

$w(x)$ is a "weight" function, which provides some flexibility in measuring closeness. In all cases consider $w(x)$ continuous and non-negative on (a,b), $\int_a^b w(x)dx$ exists and $\int_a^b w(x)dx > 0$.

Remark: A sequence of functions $\{g_k(x)\}_{k=1}^{\infty}$ is said to converge to $g(x)$ with regards to a given norm $||\cdot|| \iff$

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

□

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 [4].

def: Let $Z(x) \equiv 0$ (zero function) $\quad x \in [a, b]$.

Consider $x \in [a, b] \quad a = 0, b = 3$ in what follows.

We are going to propose a 1-parameter family of functions $f_k(x)$ as candidate approximating functions to $Z(x)$:

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

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

here $k \geq 1$. See Figure 27 for an illustration.

We obtain the following from these norms:

$$||Z - f_k||_1 = \frac{1}{k}$$
$$||Z - f_k||_2 = \frac{\sqrt{2}}{\sqrt{3}}$$
$$||Z - f_k||_\infty = k.$$

So, the L_1-norm drops as k increases, the L_2-norm stays constant, the L_∞-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

$$\lim_{k\to\infty} ||Z - f_k||_1 = 0$$
$$\lim_{k\to\infty} ||Z - f_k||_2 = \sqrt{2/3}$$
$$\lim_{k\to\infty} ||Z - f_k||_\infty = \infty$$

[4] I learned this example in my own class-taking; I can neither claim it is mine, nor can I trace its origins.

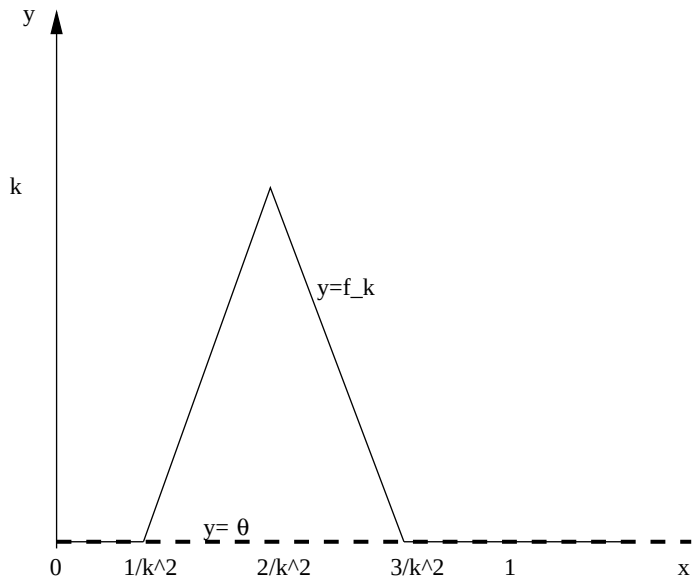

Figure 27: Approximation of $Z(x)$ with $f_k(x)$.

therefore the sequence $\{f_k(x)\}_{k=1}^{\infty}$ converges to $Z(x)$ 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 $f = Z(x)$ 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 THESE 3 NORMS:

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

- $$\begin{aligned} ||f-q||_1 &= \int_a^b |f(x)-g(x)|w(x)dx \leq \max_{a\leq x\leq b}|f(x)-g(x)| \int_b^a w(x)dx \\ &= ||f-q||_a \left(\int_a^b w(x)dx\right) \end{aligned}$$

$$\text{or } ||\cdot||_1 \leq ||\cdot||_\infty \underbrace{\int_a^b w(x)dx}_{\text{constant}}$$

- $$||f-q||_2 \leq ||f-q||_\infty \left(\int_a^b w dx\right)^{\frac{1}{2}}$$
$$\text{or } ||\cdot||_2 \leq ||\cdot||_\infty \left(\int_a^b w(x)dx\right)^{\frac{1}{2}}$$

- 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.

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

7 INTERPOLATION

7.1 Preliminaries

Problem: suppose we are given $n+1$ pairs (x, y)

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

$n+1$ values

and we want to find a polynomial p of lowest degree for which

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

Here, p is then interpolating polynomial of the data.

Theorem I: 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

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

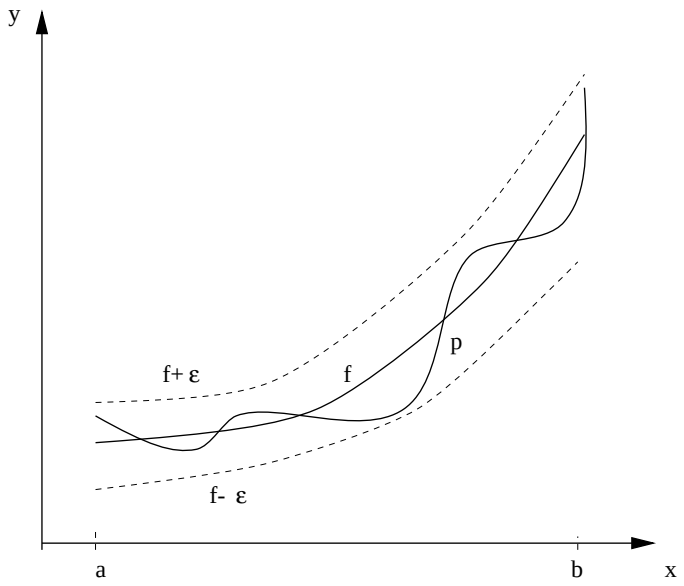

Figure 28: Illustration of Weierstrass' Theorem.

where $p_n(x) = a_0 + a_1x^1 + \cdots a_nx^n \quad n \geq 0$

□ Why interpolate with polynomials? They approximate continuous functions uniformly. That is, given any function, defined and continuous on closed interval, there exists a polynomial that is as *close* to the given function as desired in the infinity norm (the L_∞-norm).

Weierstrass Approximation Theorem: If f is defined and continuous on $[a, b]$ and $\varepsilon > 0$ is given, $\Rightarrow \exists$ polynomials p_ε, defined on $[a, b]$ with the property

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

The situation is shown in Figure 28

□

Remark: $g(x) = f(a) + f'(a)(x - a) + \cdots + \frac{f^{(k)}}{k!}(x - a)^k$ 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

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

□

Example 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 tat $q'(\alpha) \sim f'(x)$ 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.

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 amd $f(x) \in C[a, b]$.

So how close are $f(x)$ and $g(x)$? We see that $f(x) - g(x)$ is "small" in some sense. But how do we measure closeness? We develop some sort of measure of size for functions in $C[a, b]$:

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 $p(x) \in p_n(x)$

□

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 preceeding 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

Again, the problem is to interpolate $f(x)$ by $p(x) \in \mathcal{P}_n$ in $C[a,b]$

Def $\mathcal{P}_n \equiv$ set of all polynomials of degree n or less. For example,

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

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 $\{x_j\}_{j=0}^n$ be $(n+1)$ distinct points in $[a,b]$. Let$\{y_j\}_{j=0}^n$ 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

$$\ell_j(x) \equiv \frac{(x-x_0)(x-x_1)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_i)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)}$$
$$\equiv \prod_{\substack{i=0 \\ i\neq j}}^{n} \frac{(x-x_i)}{(x_j-x_i)}$$

These are called *Cardinal Functions*.

Def The Kronecker Delta Function δ_{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

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

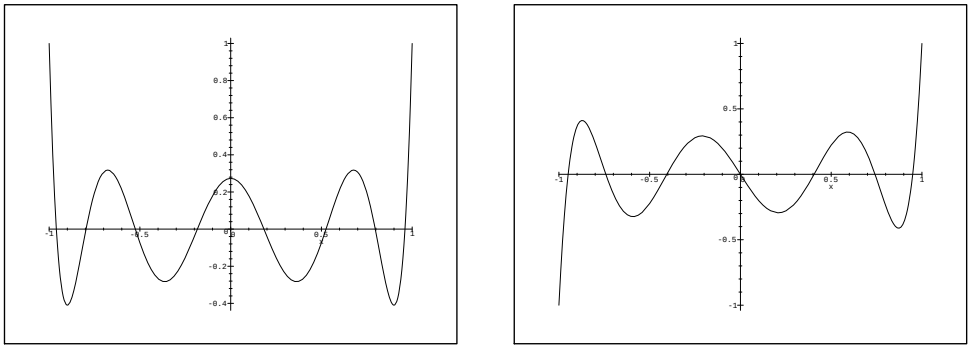

Figure 29: Illustration of $\ell_j(x)$, for $x \in [-1, 1]$, for (a) n even, (b) n odd.

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 29, for x_i equidistant.

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

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

that belongs to $\mathcal{P}_n$.

(Uniqueness): assume there are two different polynomials $p(x) \in \mathcal{P}_n$, $q(x) \in \mathcal{P}_n$ such that $p(x_j) = q(x_j) = y_i$ for $0 \leq j \leq n$. If $r(x) = p(x) - q(x)$ then $r(x) \in \mathcal{P}_n$ 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.

□

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

equations which can uniquely determine the constants. It will show up in integration problems (see 9), and the ODE section (see the ODE) section, etc.

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

$$p(x_j) = a_0 + a_1 x_j + a_2 x_j^2 + \cdots a_n x_j^n \quad 0 \leq j \leq n \tag{11}$$

In matrix form

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & : & : & & \\ : & : & : & & \\ 1 & x_n & x_n^2 & & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ : \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ : \\ y_n \end{bmatrix} \tag{12}$$

or $V\mathbf{a} = \mathbf{y}$ 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 $\mathbf{0}$ is the only solution of $V\mathbf{a} = \mathbf{0}$ (or determinant of $V \neq 0$).

So if $\mathbf{a}$ is any solution of $V\mathbf{a} = \mathbf{0}$ then $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ is an n^{th} degree polynomial with $p(x_j) = 0, \quad j = 0, 1 \cdots, n$. Since the only n^{th} degree polynomial with $(n+1)$ zeros is the zero polynomial, it must be that $\mathbf{a} = 0$ therefore V is non-singular. Thus (11) must have a unique solution and the polynomial $p(x)$ is found by solving (12), and is the unique interpolating polynomial in $\mathcal{P}_n$.

Remarks

(1) if $f(x) \in \mathcal{P}_n$ then $f(x) = p(x) \, \forall x$ by uniqueness property.

(2) The solution of the Vandermonde matrix problem, (12), 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 $q(x) \in \mathcal{P}_n$ which satisfy $q(x_j) = y_j \; 0 \leq j \leq n$.

Example Suppose we want a second degree polynomial $p(x)$ such that

x	0	1	2
p	-1	2	7.

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 $p(x) = -\ell_0(x) + 2\ell(x) + 7\ell_2(x) = x^2 + 2x - 1$

Using the Method of Undetermined coefficients: since $x_0 = 0$, $x_1 = 1$, $x_2 = 2$: Let

$$\begin{aligned} p(x_j) &= a_0 + a_1 x + a_2 x^2 \\ \\ p(0) &= a_0 \quad = -1 \\ p(1) &= a_0 + a_1 + a_2 = 2 \\ p(2) &= a_0 + 2a_1 + 4a_2 = 7. \end{aligned}$$

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

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

□

Remark Often the y_j's are determined by some function $f(x)$ so we have $p(x_j) = f(x_j), \quad 0 \le j \le n.$

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

$$\ell_0(x) = \frac{(x-2.5)(x-4)}{(2-2.5)(2-4)} = (x-6.5)x + 10$$
$$\ell_1(x) = \frac{(x-2)(x-4)}{(2.5-2)(2.5-4)} = \frac{(-4x+24)x - 32}{3}$$
$$\ell_2(x) = \frac{(x-2)(x-2.5)}{(4-2)(4-2.5)} = \frac{(x-4.5)x + 5}{3}$$

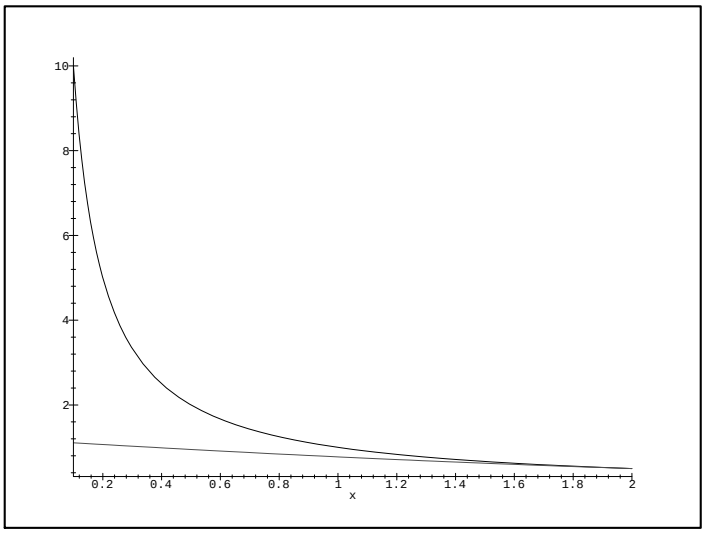

Figure 30: Example of a function $f(x) = 1/x$ and its 2^{nd} degree interpolation $p(x) = (0.05x - 0.425)x + 1.15$.

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 30

Theorem: If $x_0, x_1, \cdots x_n$ are distinct numbers and $f \in C^{n+1}[a, b]$ then for each $x \in [a, b]$ and 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.

□

The Error in 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 $x_0, x_1 \cdots x_n \in [a,b]$. For each $x \in [a,b]$ there corresponds a point $\xi \in (a,b)$ such that

$$f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi(x)) \prod_{i=0}^{n} (x - x_i), \tag{13}$$

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

Proof: Let $R \equiv f - p(x)$. Since (13) is trivially satisfied at $x_0, x_1, ...x_n$, we need only to focuse 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 $g : [a,b] \to \mathcal{R}$, the whole real line, with

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

with $z \in [a,b]$. By assumption of f, the function Q is $n+1$-times continuously differentiable, and by construction, has $n+1$ zeros, at the locations $x_0, x_1, ...$ 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 ξ. For this zero we have that

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

and from this we obtain (13).

□

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

$$\begin{aligned}
l_0(x) &= \frac{(x+\frac{\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3})}{(x_0+\frac{\pi}{3})(x_0-\frac{\pi}{3})(x_0-\frac{2\pi}{3})} \\
&= \frac{-9(x+\frac{\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3})}{4\pi^3} \\
l_1(x) &= \frac{(x+\frac{2\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3})}{(x_1+\frac{2\pi}{3})(x_1-\frac{\pi}{3})(x_1-\frac{2\pi}{3})} \\
&= \frac{9(x+\frac{2\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3})}{2\pi^3} \\
l_2(x) &= \frac{(x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{2\pi}{3})}{(x_2+\frac{2\pi}{3})(x_2+\frac{\pi}{3})(x_2-\frac{2\pi}{3})} \\
&= \frac{-9(x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{2\pi}{3})}{2\pi^3} \\
l_3(x) &= \frac{(x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{\pi}{3})}{(x_3+\frac{2\pi}{3})(x_3+\frac{\pi}{3})(x_3-\frac{\pi}{3})} \\
&= \frac{9(x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{\pi}{3})}{4\pi^3}
\end{aligned}$$

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

$$\begin{aligned}
p(x) &= \frac{-\sqrt{3}}{2}l_0(x) - \frac{\sqrt{3}}{2}l_1(x) + \frac{\sqrt{3}}{2}l_2(x) + \frac{\sqrt{3}}{2}l_3(x) \\
&= \frac{9\sqrt{3}}{8\pi^3}\Big((x+\frac{\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3}) - 2(x+\frac{2\pi}{3})(x-\frac{\pi}{3})(x-\frac{2\pi}{3}) \\
&\quad -2(x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{2\pi}{3}) + (x+\frac{2\pi}{3})(x+\frac{\pi}{3})(x-\frac{\pi}{3})\Big)
\end{aligned}$$

Figure 31 shows the interpolating function as well as $sin(x)$ the function which we are approximating. Equation 13 tells us that the error for this function is equal to

$$\begin{aligned}
f(x) - p(x) &= \frac{1}{4!}f^{(4)}(\xi(x))\prod_{i=0}^{3}(x-x_i) \\
&= \frac{1}{24}\sin(\xi(x))\prod_{i=0}^{3}(x-x_i)
\end{aligned}$$

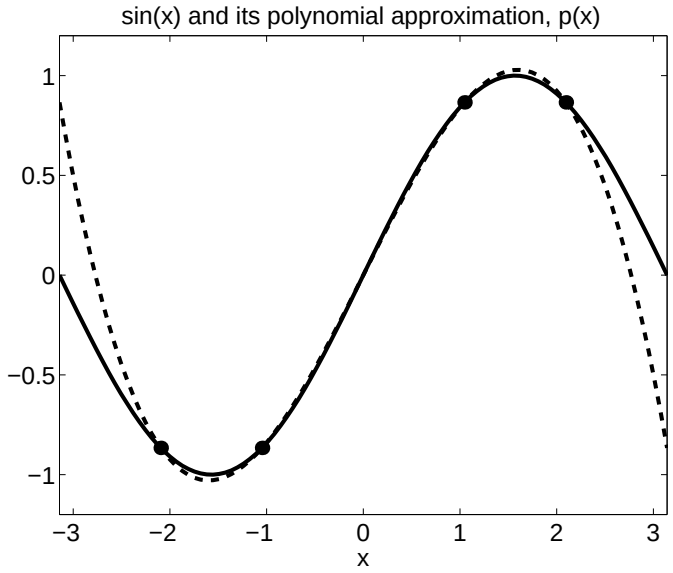

Figure 31: Plot of function $\sin(x)$ (solid line) and its interpolating polynomial $p(x)$ (dashed line) for the points $x_0 = \frac{-2\pi}{3}$, $x_1 = \frac{-\pi}{3}$, $x_2 = \frac{\pi}{3}$ and $x_3 = \frac{2\pi}{3}$. The dots show the interpolation points of the polynomial approximation.

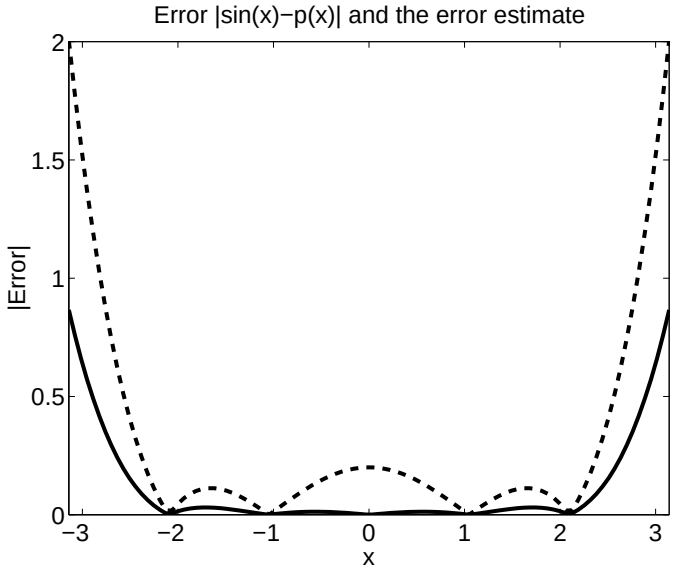

Figure 32: Plot of function the error $|\sin(x) - p(x)|$ (solid line) and also the estimate for the error, $\frac{1}{24}|\prod_{i=0}^{n}(x - x_i)|$ (dashed line). Note that the only points where the estimate of the error agrees with the error is at the interpolation points where we know it is zero.

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 less than $\frac{1}{24}|\prod_{i=0}^{n}(x - x_i)|$. Figure 32 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:

```
Enter numbers (x0,x1, x2,.....xn)
evaluate f(x0), f(x1), f(x2),...f(xn) as the first colum S(0,0) S(1,0)
S(2,0)....S(n,0) of S.

Output table S with p(x) = S(n,n)
```

```
for i = 1, 2,...n
    for j = 1, 2, ....,i
        S(i,j) =
        [(x - x(i-j)) S(i,j-1) - (x - xi)S(i-1,j-1)]/[xi - x(i - j)]
    end
end
Output (S);
stop
```

Here is a matlab code that implements this. **Note that this code generates the function value corresponding to the Lagrange polynomial**.

7.3 Chebyshev Polynomials

Chebyshev Polynomials are orthogonal polynomials with respect to a weight over the interval -1 to 1 and obey a recursion relation. We can use them in the following way: there's a term in (13) that can be optimized thus reducing the error.

Chebyshev polynomials obey the relations

$$\begin{cases} T_0(x) = 1 & T_1(x) = x \\ T_{n+1}(x) = & 2xT_n(x) - T_{(n-1)}(x) \quad n \geq 1 \end{cases}$$

The first four Chebychev polynomials appear in Figure 33

Here are first six polynomials:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ T_5(x) &= 16x^5 - 20x^3 + 5x \end{aligned}$$

□

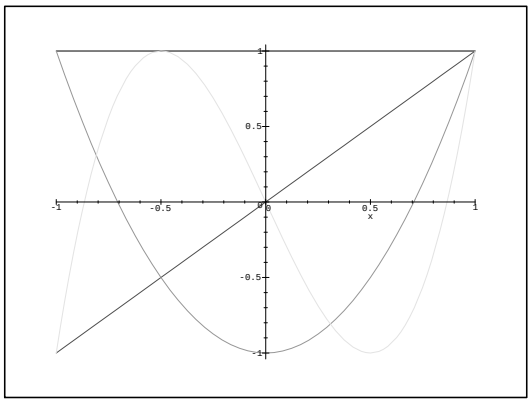

Figure 33: Illustration of T_0, T_1, T_2, and T_3.

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. $T_n(\cos(\theta)) = 0, \quad \theta = \frac{(2k-1)\pi}{2n}, \qquad k = 1, 2, ..., n$
5. $T_n(\cos(\theta)) = (-1)^k, \quad \theta = \frac{k\pi}{n}, \qquad k = 0, 1, 2..., n$
6. $T_{2n+1}(0) = 0$
7. There are n roots and these are distinct.
8. Polynomial $\dfrac{T_n(x)}{2^{n-1}}$ has leading coefficient of 1 i.e. it's a "monic polynomial" and

$$\max_{-1 \leq x \leq 1} \left| \frac{1}{2^{n-1}} T_n(x) \right| = \frac{1}{2^{n-1}} \tag{14}$$

□

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 $[a, b] \to [-1, 1]$. 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 $x \in [a, b]$

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

□

Supposing we are interpolating a sufficiently smooth function $f(x)$ (scaled to the interval $[-1, 1]$). Suppose that the function then defines the data points to interpolate on $-1 < x_0 < x_1 ... < x_n < 1$. If we used Lagrange polynomials, we obtain the sup-norm error

$$(15)\ ||f(x) - p(x)||_\infty \leq \frac{1}{(n+1)} \max_{|x|\leq 1} \left| f^{(n+1)}(\xi(x))| \max_{|x|\leq 1} | \prod_{i=0}^{n} (x - x_i) \right|.$$

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

$$\max_{|x|\leq 1} | \prod_{i=0}^{n} (x - x_i)| \text{ is minimized,}$$

where $\prod_{i=0}^{n} (x - x_i)$ is an $(n+1)^{th}$ order polynomial.

This is in fact possible, as we shall see.

Let x_i be the zeros of the $\frac{1}{2^n} T_{n+1}(x)$ polynomial.

$$x_i = \cos\left(\frac{(2j+1}{2n}\pi\right) \qquad 0 \leq j \leq n.$$

Note, by the way that

$$x_j = \cos\frac{j\pi}{n+1}, \qquad j = 0, 1, 2, \ldots, n+1$$

gives the extremas.

Anyway, (15) becomes

$$||f(x) - p(x)||_\infty \leq \frac{1}{(n+1)!}\frac{1}{2^n}\max\left|f^{(n+1)}(\xi(x))\right|.$$

Statement (14) 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 $V\left(\cos(\frac{i\pi}{n+1})\right) < \prod_{i=0}^{n}(x - \cos(\frac{i\pi}{n+1})) = \frac{1}{2^n}$ for $0 \leq i \leq n+1$. Or else, $||V||_\infty > ||\Pi(x - x_i)||_\infty$.

If i is odd then $V\left(\cos\frac{i\pi}{n+1}\right) > \prod_{i=0}^{n}\left(x - \cos(\frac{i\pi}{n+1})\right) = -\frac{1}{2^n}$, or again,

$||V||_\infty \geq ||\Pi(x - \cos\left(\frac{i\pi}{n+1}\right))||_\infty$. 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 $-1 \leq x \leq 1$.

□

This error formula is important because $\ell_j(x)$ are used extensively in deriving numerical differentiation and integration schemes and the same arguments are used there.

□

7.4 Newton's Divided Differences

Expressing a polynomial as

$$p(x) = y_0 l_0(x) + y_1 l_1(x) \ldots + y_n l_n(x)$$

is cumbersome and somewhat inefficient. A more convenient structure is

$$\begin{aligned} p(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_0)(x - x_1)(x - x_2) + \\ (16) \quad & \cdots a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}), \end{aligned}$$

which is known as the *Newton form*. Here the a_i, $i = 1..n$ are constants.

As before

$$p(x_0) = f(x_0), p(x_1) = f(x_1) \ldots, p(x_n) = f(x_n). \tag{17}$$

The a's in (16) are found using (17): the first two,

$$\begin{aligned} p(x_0) = a_0 &= f(x_0) \\ p(x_1) = f(x_0) + a_1(x_1 - x_0) &= f(x_1) \end{aligned}$$

Therefore

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

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

$$f[x_i, x_{i+1}, x_{i+2} \cdots x_{i+k-1}] \text{ and } f[x_{i+1}, x_{i+2}, \cdots x_{i+k-1}, x_{i+k}].$$

Hence, the k^{th} divided difference satisfies

$$f[x_i, x_{i+1}, x_{i+2} \ldots x_{i+k-1}, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2} \ldots x_{i+k}] - f[x_i, x_{i+1}, \cdots x_{i+k-1}]}{x_{i+k} - x_i}$$

and the $a_k = f[x_0, x_1, x_2 \cdots x_k]$.

For example

$$\begin{aligned} f[x_2, x_3] &= \frac{f[x_3] - f[x_2]}{x_3 - x_2} \\ f[x_3, x_4, x_5] &= \frac{f[x_4, x_5] - f[x_3, x_4]}{x_5 - x_3}. \end{aligned}$$

Hence, we can get

$$\begin{aligned} a_1 &= f[x_0, x_1] \\ a_2 &= f[x_0, x_1, x_2] \end{aligned}$$

with a little algebra, and so on.

With this notation we can write down (16) as

$$p(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1 \cdots x_k](x - x_0) \cdots (x - x_{k-1}).$$

In the algorithm that we present below, we use the notation

$$F_{i,1} = f[x_i, x_{i+1}].$$

We define

$$F_{i,j} \equiv \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}.$$

Note that $F_{i,i} = f[x_0, x_1 \ldots x_i]$.

Algorithm: input $x_0, x_1 \cdots x_n$ distinct, $f(x_0), f(x_1) \cdots f(x_n)$

output $F_{0,0}, F_{1,1}, \ldots, F_{n,n}$ where

$$p(x) = \sum_{i=0}^{n} F_{i,i} \Pi_{j=0}^{i-1}(x - x_j)$$

note $f(x_0) \equiv F_{0,0}; f(x_i) = F_{1,0}; f(x_2) = F_{2,0} \cdots f(x_n) = F_{n,0}$

Step 1 for $i = 1, 2 \ldots n$
for $j = 1, 2 \ldots i$
$$F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$$

Step 2 Output $(F_{0,0}; F_{1,1} \ldots F_{n,n})$
End

x	$f(x)$	1st div diff	2nd div diff	3rd div diff
x_0	$f[x_0]$			
		$f[x_0, x_1] = \frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_1	$f[x_1]$		$f[x_0, x_1, x_2] = \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
		$f[x_2, x_1] = \frac{f[x_2]-f[x_1]}{x_2-x_1}$		$f[x_0, x_1, x_2, x_3] = \frac{f[x_1,x_2,x_3]-f[x_0,x}{x_3-x_0}$
$\vdots$	$\vdots$	$\vdots$	$f[x_1, x_2, x_3] = \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	
	$f[x_4]$			
$\vdots$		$f[x_4, x_5] = \frac{f(x_5]-f[x_4]}{x_5-x_4}$		
x_5	$f[x_5]$			

$f(x) = x^6 - 2x^5 - 4x^4 + x^3 + x - 20$

x	$f(x)$
-3	841
-2	34
-1	-23
0	-20
1	-23
2	-74
3	-71
4	1072

8 points thus 8 divided difference levels:

x	0 DD $f[x]$	1 DD $f[x,x]$	2 DD $f[x,x,x]$	$\cdots$ $\cdots$	$\cdots$ $\cdots$	$\cdots$ $\cdots$	$\cdots$ $\cdots$	7 DD $f[x,x,x,x,x,x,x,x]$
-3	841							
		-807						
-2	34		375					
		-57		-115				
-1	-23		30		26			
		3		-11		-5		
0	-20		-3		1		1	
		-3		-7		1		0
1	-23		-24		6		1	
		-51		17		7		
2	-74		27		41			
		3		181				
3	-71		570					
		1143						
4	1072							

□

7.4.1 Interpolation at Equally-Spaced Points

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

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{h}$$
$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{2h^2}$$

We define the *forward difference* operator Δ. 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

$$\begin{aligned} \Delta^2 f &= \Delta(\Delta f) = \Delta(f(x+h) - f(x)) \\ &= (f(x+2h) - f(x+h)) - (f(x+h) - f(x)) \\ &= f(x+2h) - 2f(x+h) + f(x) \end{aligned}$$

thus

$$\Delta^k f = \sum_{i=0}^{k} \binom{k}{i} (-1)^i f(x + (k-i)h)$$

where

$$\begin{aligned}\binom{k}{i} &= \frac{k!}{i!(k-i)!}\\ \binom{k}{0} &= \binom{k}{k} = 1.\end{aligned}$$

Recall

$p(x) = f(x_0)+f[x_0,x_1](x-x_0)+\cdots f[x_0,x_1,\cdots,x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})$

change of variable

$$x = x_0 + \rho h \quad \rho \quad \text{is a number}$$

then

$$[x_0, x_1 \cdots x_j](x-x_0)(x-x_1)\cdots(x-x_{j-1}) = \frac{\Delta^j f(x_0)}{j!} \underbrace{\rho(\rho-1)(\rho-2)\cdots(\rho-(j-1))}_{\substack{\text{polynomial of degree } j \\ \text{in the variable } \rho}}.$$

Note that

$$\binom{r}{j} = \frac{r(r-1)(r-2)\cdots(r-(j-1))}{j!}.$$

So, for $j \geq 1$

$$\binom{r}{0} = 1$$

$$\binom{r}{1} = \alpha \quad \binom{r}{2} = \frac{r(r-1)}{2}\binom{r}{3} = \frac{r(r-1)(r-2)}{6},$$

etc.

So we can write

$$p(x) = p(x_0+\rho h) = f(x_0) + \Delta f(x_0)\binom{\rho}{1} + \Delta^2 f(x_0)\binom{\rho}{2}\cdots\Delta^n f(x_0)\binom{\rho}{h}$$

Hence,

$$p(x_0+\rho h) = \sum_{i=0}^{n} \Delta^i f(x_0)\binom{\rho}{i}.$$

□

Newton's forward Formula: appears tabulated in old books on functions and series and books dedicated on divided differences:

x	$f(x)$	Δf	$\Delta^2 f$	$\cdots$	$\Delta^n f$
x_0	$f(x_0)$				
		$\Delta f(x_0)$			
x_1	$f(x_1)$		$\Delta^2 f(x_0)$		
		$\Delta(x_1)$			
x_2	$f(x_2)$		$\Delta^2 f(x_4)$		
x_3	:	:	:	$\cdots$	$\Delta^n f(x_0)$
:	:	:	$\Delta^2 f(x_{n-2})$		
x_{n-1}	:	$\Delta f(x_{n-1})$			
x_n	$f(x_n)$				

□

7.5 Remarks on Polynomial Interpolation

The problem with polynomial interpolation? it seems overkill to use a 99^{th} degree polynomial to fit 100 data points. But the main problem is this: high degree polynomials are very smooth but are usually very oscillatory. This is particularly troublesome if the data comes from an experiment, which means that the data is very noisy. In the homework you get to explore how high degree polynomial interpolation can lead to unusual results.

A radically different approach, much favored now in modern computing, is to use low order polynomials and control the error by making the grid very fine. This, of course, assumes that you have control of generating the data points.

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. The splines are made of lead, coted with plastic. They use these to connect the points on a picture, effectively interpolating between the "knots" or fixed points in a nice smooth way, as will cubic polynomials can appear to the eye.

Later on we'll consider "trigonometric interpolation" which uses trigonometric functions which are infinitely smooth.

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 $[x_i, x_{i+i}]$, $i = 0, 1, 2, \cdots, n -$

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 $|\Delta| \equiv \max_{1 \le i \le n-1} \Delta x_i$, 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

$$\mathcal{S}_m^k(\Delta) = \left\{ s : s \in C^k[a,b], s\Big|_{[x_i, x_{i+1}]} \in \mathcal{P}_m, i = 1, 2, \cdots, n-1 \right\} \tag{18}$$

where $m \ge 0, k \ge 0$. $\mathcal{S}_m^k(D)$ the "spline functions of degree m and smoothness class k" relative to the subdivision D.

The continuity assumption of (18) 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 $s \in \mathcal{S}_m^m$ 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 $s \in \mathcal{S}_1^0(D)$ such that for f on $[a,b]$

$$s(x_i) = f_i; \quad \text{where } f_i = f(x_i), \quad i = 1, 2, \ldots, n.$$

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 $x_i \le x \le x_{i+1}$, $i = 1, 2, 3, \ldots, n-1$. This is shown schematically in Figure 34.

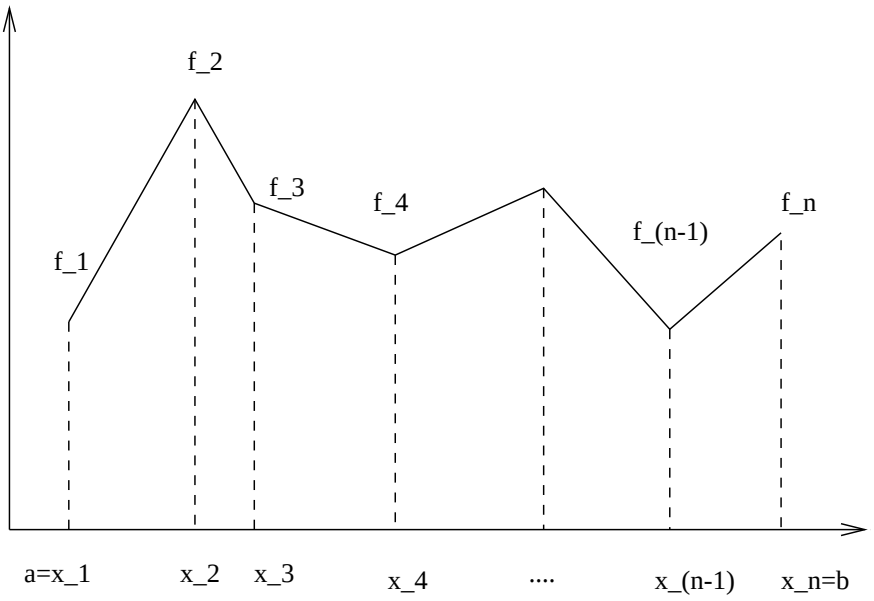

Figure 34: Discretization of the domain.

What is the error? First recall from polynomial interpolation that for $x \in [x_0, x_1]$ the error for polynomial interpolaters 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

$$f(x) - s_1(f; x) = (x - x_i)(x - x_{i+1}) f[x_i, x_{i+1}, x], \qquad x \in [x_i, x_{i+1}]$$

$$|f(x) - s_1(f;x)| \leq \frac{(\Delta x_i)^2}{8} \max_{x \in [x_i, x_{i+1}]} |f''|, \quad \text{with } x \in [x_i, x_{i+1}]$$

$$\text{thus} \quad ||f(x) - s_1(f;x)||_\infty \leq \frac{1}{8}|\Delta|^2 ||f''||_\infty$$

Which shows that the error can be made arbitrarily be make arbitrarily small by choosing the Δx_i's small: the cost is the volume of data.

A Basis for $\mathcal{S}_1^\circ(\Delta)$: How many degrees of freedom do we have? We have 2 per segment and there are $n-1$ segments. Hence

$$\dim \mathcal{S}_1^{-1}(D) = 2n - 2$$

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

$$B_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if} \quad x_{i-1} \leq x \leq x_i \\ \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if} \quad x_i \leq x \leq x_{i+1} \\ \\ 0 & \text{otherwise} \end{cases}$$

We ignore 1^{st} equation at $i = 1$ and ignore 2^{nd} equation at $i = n$.

These B_i's are called "hat" functions, and an illustration of them appear in Figure 35

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

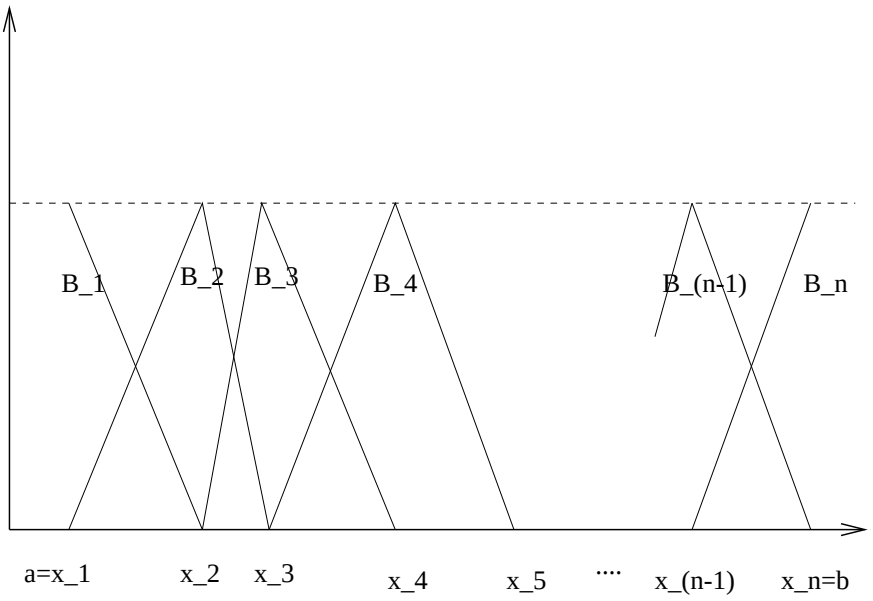

Figure 35: The "hat" functions $B_i(x)$.

(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 $x_i \quad i = 1, 2, 3, \ldots, n$, 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 $\widehat{s}_1(f; \cdot) \in \mathcal{S}_1^\circ(D)$ such that

$$\int_a^b |f(x) - s_1(f; x)|^2 dx \leq \int [f(x) - s(x)]^2 dx \quad \forall s \in \mathcal{S}_1^\circ(\Delta) \tag{19}$$

Writing $\widehat{s}(f; x) = \sum_{i=1}^{n} \widehat{c}_i B_i(x)$ from (19)

$$\begin{aligned} \widehat{s}_1(f; x) - f(x) &= \text{E} \\ \sum_{i=1}^{n} \widehat{c}_i B_i(x) - f(x) &= \text{E} \end{aligned}$$

where E is the error. Next, multiply both sides by $B_j(x)$ and integrate:

$$\sum_{i=1}^{n} \left[\int_a^b B_i(x) B_j(x) dx \right] \widehat{c}_i \approx \int_a^b B_i(x) f(x) dx \quad i = 1, 2, 3, \ldots, n. \tag{20}$$

$$B_i \text{ is nonzero on } (x_i, x_{i+1}) \Rightarrow \int_a^b B_i(x) B_j(x) dx = 0 \quad \text{if } |i - j| > 1.$$

The resulting system of equations generates a matrix which is tridiagonal. An easy calculation (integrate (20)) shows that

$$\begin{aligned} &\frac{1}{6}\Delta x_{i-1}\widehat{c}_{i-1} + \frac{1}{3}(\Delta x_{i-1} + \Delta x_i)\widehat{c}_i + \frac{1}{6}\Delta x_i \widehat{c}_{i+1} = b_i; \\ &i = 1, 2, 3, \cdots, n \end{aligned} \tag{21}$$

Note that $\Delta x_0 = 0 \quad \Delta x_n = 0$, by convention.

The matrix problem in (21) is symmetric positive definite and easy to solve using the algorithm (12.2.1).

□

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 piecewise 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 $Sp_n^k(x)$ 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:

$$\begin{cases} S_n^3(x) \in C^2[a,b] \text{ i.e. } S_n^3(x), \partial_x S_n^3(x), \partial_x^2 S_n^3(x) \text{ continuous on } [a,b] \\ S_n^3(x_j) = f(x_j) \equiv f_j \quad 0 \le j \le n \qquad \text{i.e.} S_n^3(x) \quad \text{interpolates } f(x) \text{ on } [a,b] \\ S_n^3(x) \text{ is a cubic polynomial on each interval} \quad [x_j, x_{j+1}] \quad 0 \le j \le n-1. \end{cases} \tag{22}$$

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 $S_j^3(x) \in \mathcal{P}_3$ it has 4 coefficients. However, we would like it to satisfy (22) therefore we have 12 coefficients to pin down (count them).

Since each $S_j^3(x) \in \mathcal{P}_3$, 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

$$\begin{cases} S_0^3(x_0) = f_0 & S_1^3(x_1) = f_1 & S_2^3(x_2) = f_2 & S_2^3(x_3) = f_3 \\ S_0^3(x_1) = S_1^3(x_1) & S_1^3(x_2) = S_2^3(x_2). \end{cases}$$

Since $S_n^3(x)$ and its first two derivatives are continuous at x_1 and x_2 we can get four more equations:

$$\begin{cases} \partial_x S_0^3(x_1) = \partial_x S_1(x_1) & , \quad \partial_x^2 S_0^3(x_1) = \partial_x^2 S_1^3(x_1) \\ \partial_x S_1^3(x_2) = \partial_x S_2^3(x_2) & , \quad \partial_x^2 S_1^3(x_2) = \partial_x^2 S_2^3(x_2) \end{cases}$$

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

$$\begin{aligned}\partial_x^2 S^3(x_0) &= \partial_x^2 S^3(x_3) = 0 \\ &\quad \text{(“free” or “natural” boundary condition).} \\ \partial_x S^3(x_0) &= \partial_x f_0 \text{ and } \partial_x S^3(x_3) = \partial_x f_3 \\ &\quad \text{(“clamped” boundary condition).}\end{aligned}$$

Generalizing, the complete list of properties is

$$S_j^3 \text{ is a cubic polynomial on } [x_j, x_{j+1}] \quad j = 0 \cdots n-1. \tag{23}$$

$$S^3(x_j) = f_j \quad j = 0 \cdots n \tag{24}$$

$$S_{j+1}^3(x_{j+1}) = S_j^3(x_{j+1}) \quad j = 0 \cdots n-2 \tag{25}$$

$$\partial_x S_{j+1}^3(x_{j+1}) = \partial_x S_j^3(x_{j+1}) \quad j = 0 \cdots n-2 \tag{26}$$

$$\partial_x^2 S_{j+1}^3(x_{j+1}) = \partial_x^2 S_j^3(x_{j+1}) \quad j = 0 \cdots n-2, \tag{27}$$

plus one of the following is satisfied:

(i) $\partial_x^2 S^3(x_0) = \partial_x^2 S^3(x_n) = 0$ 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

$$S_j^3(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3, \tag{28}$$

to construct the splines. Use $h_j = x_{j+1} - x_j$, with $j = 0, ..., n-1$, and solve for $\mathbf{c} \equiv [c_0, c_1, ..., c_n]$. Show that for the *natural end conditions*, i.e. $\partial_x^2 S^3(a) = \partial_x^2 S^3(b) = 0$, the full family of splines is given by the solution to the system

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

A is $(n+1)\times(n+1)$ matrix given by

$$A=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{c}=\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} \qquad \mathbf{g}=\begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2-a_1)-\frac{3}{h_0}(a_1-a_0) \\ \vdots \\ \vdots \\ \frac{3}{h_{n-1}}(a_n-a_{n-1})-\frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\ 0 \end{bmatrix}$$

Similarly, for the *clamped case*, $\partial_x S^3(a)=\partial_x f(a)$ and $\partial_x S^3(b)=\partial_x f(b)$,

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

is explicitly given by

$$A=\begin{bmatrix} 2h_0 & h_0 & 0 & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & 0 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$

$$\mathbf{c}=\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ \vdots \\ c_n \end{bmatrix} \qquad \mathbf{g}=\begin{bmatrix} \frac{3}{h_0}(a_1-a_0)-3f'(a) \\ \frac{3}{h_1}(a_2-a_1)-\frac{3}{h_0}(a_1-a_0) \\ \vdots \\ \vdots \\ \frac{3}{h_{n-1}}(a_n-a_{n-1})-\frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\ 3f'(b)-\frac{3}{h_{n-1}}(a_n-a_{n-1}) \end{bmatrix}$$

□

Algorithm Natural Cubic Spline

input $n; x_0 \cdots x_n, a_0 = f(x_0) \cdots a_n = f(x_n)$
output $a_j, b_j, c_j d_j$ for $j = 0, 1 \cdots n-1$

Step 1 for $i = 1 \cdots n-1 \quad h_i = x_{i+1} - x_1$
Step 2 for $i - 1 \cdots n - 1$

$$a_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$$

Step 3 $l_0 = 1$
$\mu_0 = 0$
$z_0 = 0$

Step 4 for $i = 1 \cdots n-1$
$l_i = 2(x_{j+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$
$\mu_i = h_i / l_i$
$z_i = (\alpha_i - h_{i-1} z_{i-1})/l_i$

Step 5 $\quad l_n = 1$
$x_n = 0$
$c_n = 0$

Step 6 $\quad$ for $j = n-1 \cdots 0$
$c_j = z_j - \mu_j c_{j+1}$
$b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$
$d_j = (c_{j+1} - c_j)/(3h_j)$

Step 7 $\quad$ output $(a_j, b_j, c_j, d_j$ for $j = 0 - n - 1)$

STOP

Clamped Cubic Spline Case

input $n, x_0 \cdots x_n, a_0 = f(x_0) \cdots a_n = f(x_n); FPO = f'(x_0) FPN = f'(x_n)$

output $(a_j, b_j, c_j, d_j$ for $j = 0 \cdots n-1)$

Step 1 for $i = 0 \cdots n-1 \quad h_i = x_{i+1} - x_i$

Step 2 Set $\alpha_0 = 3(a_1 - a_0)/h_0 - 3$ FPO
$\alpha_n = 3$ FPN $-3(a_n - a_{n-1})/h_{n-1}$

Step 3 for $i = 1, 2 \cdots n-1$

$$\alpha_i = \tfrac{3}{h_i}(a_{i+1} - a_i) - \tfrac{3}{h_{i-1}}(a_i - a_{i-1})$$

Step 4 Set $l_0 = 2h_0$
$\mu_0 = 0.5$
$z_0 = \alpha_0/l_0$

Step 5 For $i = 1, 2 \cdots n-1$
$l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$
$\mu_i = h_i/l_i$
$z_i = (\alpha_i - h_{i-1}z_{i-1})/l_i$

Step 6 $l_n = h_{n-1}(2 - \mu_{n-1})$
$z_n = (\alpha_n - h_{n-1}z_{n-1})/l_n$
$c_n = z_n$

Step 7 for $j = n-1, \ldots, 0$
set $c_j = z_j - u_j c_{j+1}$
$b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$
$d_j = (c_{j+1} - c_j)/(3hj)$

Step 8 Output $(a_j, b_j, c_j, d_j$ for $j = 0 \cdots n-1)$

STOP

□

Figures 36 and 37 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.

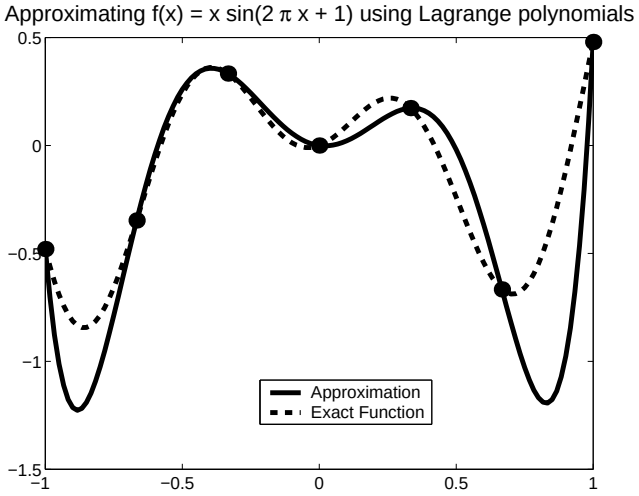

Figure 36: Approximation of $f(x) = x\sin(2\pi x + 1)$ using a sixth order Lagrange polynomial.

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

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k \cos(kwx) + b_k \sin(kwx)]$$

where $w = \dfrac{2\pi}{T}$, and the coefficients a_k's and b_k's can be found by $a_k = \dfrac{1}{T}\displaystyle\int_{-T/2}^{T/2} f(x)\cos(kwx)dx \quad b_k = \dfrac{1}{T}\int_{-T/2}^{T/2} f(x)\sin(kwx)dx$. We can also write:

$$f(x) \quad = \quad \sum_{k=-\infty}^{\infty} c_k e^{ikwx}$$

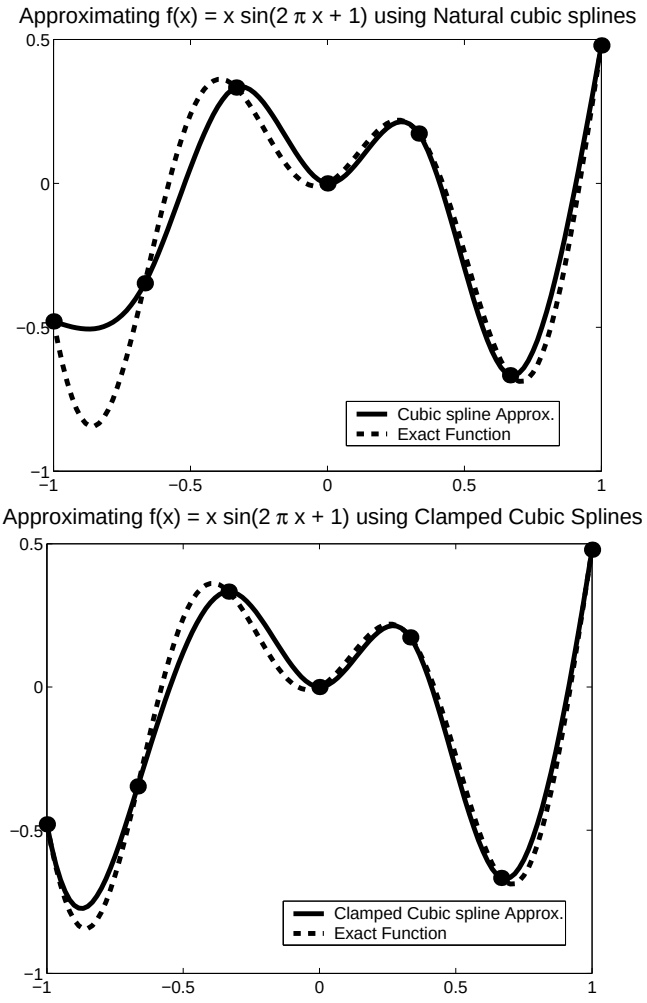

Figure 37: Approximation of $f(x) = x\sin(2\pi x + 1)$ using cubic splines. The first figure uses the natural cubic spline end conditions. The second figure uses the clamped cubic spline conditions.

$$\begin{array}{lll} \& & \text{where} \\ c_k & = & \dfrac{a_k - ib_k}{2} \text{ and } c_{-k} = \dfrac{a_k + ib_k}{2} \end{array}$$

Without loss of generality in what follows we will scale x so that the period is $T = 2\pi$ in the new variable: let

$$\widehat{f}(\tilde{x}) = f(x) \quad , \quad \tilde{x} = \frac{2\pi}{T}x.$$

Furthermore, we drop the tildes and the hats from the scaled independent and dependent variables, respectively.

Consider 2π-periodic functions which are

$$\int_{-\pi}^{\pi} |u(x)|^2 dx < \infty \quad , \text{ we say that } u \in L^2(-\pi, \pi)$$

i.e. belonging to the space of functions L^2-periodic, with period 2π. In this space the norm is given in terms of the inner product as

$$(u, v) = \int_{-\pi}^{T} \overline{u}(x)v(x)dx$$

here $\overline{u}$ is the complex conjugate of u.

Recall: if X is a linear vector space over $\mathbb{C}$, then a map

$$u, v \in X \to (u, v) \in \mathbb{C}$$

such that

(i) $(u, v) = \overline{(u, v)}$ (symmetry),

(ii) $(u, \lambda v) = \lambda(u, v)$ (scaling) $\quad \lambda \in \mathcal{C}$, 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.

Recall "Cauchy Schwarz inequality:"

$$|(u, v)| \leq ||u|| \, ||v||$$

<u>definition:</u> $u, v \in L^2$ are orthogonal if

$$(u, v) = 0$$

□

Let $\phi_k(x) \equiv e^{ikx} \quad k \in \mathbb{Z}$ then $(\phi_k, \phi_j) = \begin{cases} 2\pi & \text{if } k = j \\ 0 & k \neq 0 \end{cases}$

Claim

$$u(x) = \sum_{k=-\infty}^{\infty} \widehat{u}_k \phi_k(x) \quad , \text{ Fourier series representation}$$

where

$$\widehat{u}_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-ikx} u(x) dx = \frac{1}{2\pi} (\phi_k, u)$$

are the Fourier coefficients.

<u>Note:</u> if $\sum_{k=-\infty}^{\infty} |\widehat{u}_k| < \infty$ the $u(x)$ would be continuous by Lebesgue dominated convergence.

This is not true in general for $u \in L^2$. What is true is that if

$$\sum_{k=-\infty}^{\infty} |\widehat{u}_k|^2 \; < \; \infty$$

then

$$2\pi \sum_{k=-\infty}^{\infty} |\widehat{u}_k|^2 \; = \; ||u||_{L^2}^2$$

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:

$$\text{Let } s^{n,m}(x) = \sum_{k=-m}^{n} \widehat{u}_k \phi_k(x)$$

a partial sum

$$\lim_{n,m\to\infty} s^{n,m}(x) = u(x) \text{ in } L^2$$

$$\lim_{n,m\to\infty} ||s^{n,n}(x) - u||_{L^2} = 0.$$

(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:

$$\widehat{(\partial_x u)}_k = ik\widehat{u}_k$$
$$\widehat{(\partial_x^n u)}_k = (ik)^n \widehat{u}_k$$

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:

$$\int |\partial_x u|^2 dx = 2\pi \sum_{k=\infty}^{\infty} k^2 |\widehat{u}_k|^2 < \infty.$$

If we had n derivatives in L^2 then

$$\int |\partial_x^n u|^2 dx = 2\pi \sum_{k=-\infty}^{\infty} k^{2n} |\widehat{u}_k|^2 < \infty$$

□

Suppose that further conditions are placed on the function and its derivatives. Could we get convergence in other norms?

$\sum_{k=-\infty}^{\infty} |\widehat{u}_k| < \infty$ would be really good convergence: let us further examine the case when $u \in H^1$. Hence

$$\sum_{k\in\mathbb{Z}} \frac{(1+k^2)^{1/2}}{(1+k^2)^{1/2}} |\widehat{u}_k| = \sum_{k\in\mathcal{Z}} \frac{1}{(1+k^2)^{1/2}} (1+k^2)^{1/2} |\widehat{u}_k|.$$

The term

$$\sum_{k\in\mathbb{Z}} \frac{(1+k^2)^{1/2}}{(1+k^2)^{1/2}} |\widehat{u}_k|$$

converges and

$$\sum_{k\in\mathcal{Z}} \frac{1}{(1+k^2)^{1/2}} (1+k^2)^{1/2} |\widehat{u}_k|$$

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 <u>not</u> 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 2π-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 2π periodic.

In Figure 38 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.

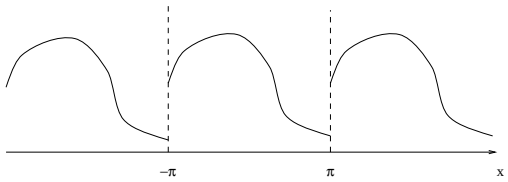

Figure 38: Extension of a function which is non-periodic (here, periodic everywhere except for the end points).

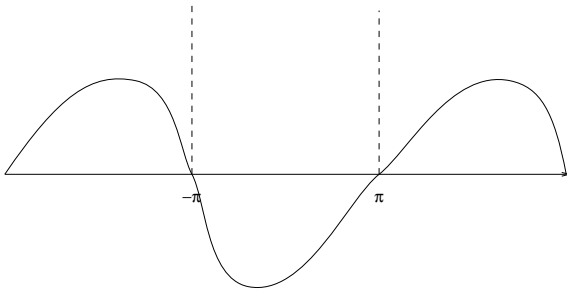

Figure 39: Extension of a non-periodic function.

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 39

   The periodic extension here is $-u(x - 2\pi)$. The resulting function now has period 2π.

B) Suppose $u(-\pi) = u(\pi) =$ 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. Again, resulting period is 4π.

D) Mixed: $u(-\pi) = 0$, $\partial_x(\pi) = 0$. In this case the period would be 8π. Exercise: construct a periodized function, with period 4π, with the above boundary conditions at $\pm\pi$.

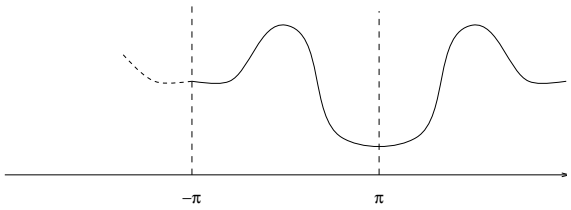

Figure 40: Extension of a non-periodic function.

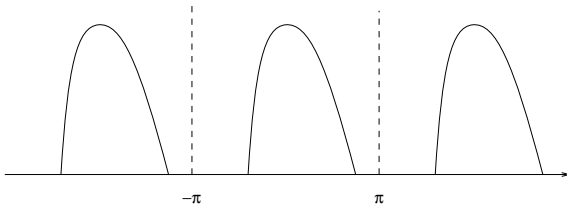

Figure 41: Extension of a non-periodic function.

E) By zero padding if the function is compactly-supported with support smaller than 2π, i.e. $u \neq 0$ in some continuous interval smaller than the period, and $u = 0$ (or a constant) otherwise. The period remains 2π. 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 2π-periodic functions (there is no difficulty considering larger or smaller T, it's just scaling):

$$f(t+2\pi) = f(t).$$

We can approximate $f(t)$ by a *trigonometric polynomial*, as:

$$p_n(t) = a_0 + \sum_{\ell=1}^{n} [a_\ell \cos(\ell\, t) + b_\ell \sin(\ell\, t)] \tag{29}$$

with $|a_n| + |b_n| \neq 0$.

For an n^{th} degree trigonometric polynomial approximation of $f(x)$ we want $f(t_\ell) = p_n(t_\ell) \quad \ell = 0, 1, 2, \ldots, 2n$ where

$$0 \leq t_0 < t_1 < t_2 \cdots < t_{2n} < 2\pi$$

and we need $2n + 1$ points in t since we're needing to fix $2n + 1$ coefficients in (29). Recall that $e^{i\theta} = \cos\theta + i \sin\theta$, hence (29) can be recast as

$$p_n(t) = \sum_{\ell=-n}^{n} c_\ell e^{i\,\ell\,t}$$

where $c_0 = a_0 \quad c_\ell = \dfrac{1}{2}(a_\ell - ib_\ell) \quad c_{-\ell} = \frac{1}{2}(a_\ell + ib_\ell)$, 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

$$p_n(z_\ell) = f(t_\ell) \quad \ell = 0, 1, \cdots, 2n. \tag{30}$$

So we take $2n+1$ distinct and equally-spaced points on the unit circle $|z| = 1$.

To see that (30) is uniquely solvable, note that (30) is equivalent to

$$Q(z_\ell) = z_\ell^n\, f(t_\ell) \quad \ell = 0, 1, \cdots 2n$$

which is a polynomial interpolation problem with $2n + 1$ distinct nodes $z_0, z_1, \cdots, z_{2n}$. 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 $\ell = 0, 1, ..., n$, hence $S(z_\ell) := 0$.

7.7.3 Evenly Spaced Grid Points

Take

$$t_\ell = \frac{2\pi}{2n+1}\,\ell \quad \ell = 0, 1, \ldots, 2n \tag{31}$$

Theorem: Let $\{t_\ell\}_{\ell=0}^{2n}$ be given by (31). Then the coefficients c_k of

$$p_n(t_\ell) \equiv \sum_{k=-n}^{n} c_k e^{ikt_\ell}$$

for $\ell = 0, 1 \cdots, 2n$ 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 $p_n(t_\ell) = f(t_\ell) \quad$ 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_ℓ} and sum over $0 \le \ell \le 2n$,

$$\sum_{\ell=0}^{2n} \sum_{m=-n}^{n} c_m e^{i(m-k)t_\ell} = \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell)$$

and interchange the order of summations on the left hand side:

$$\sum_{m=-n}^{n} c_m \underbrace{\sum_{\ell=0}^{2n} e^{i(m-k)t_\ell}}_{\text{look at this now}} = \sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell). \tag{32}$$

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 $i(m-k)t_\ell = \dfrac{i\ell(m-k)2\pi}{2n+1}$.

Let $r = e^{i(m-k)2\pi/(2n+1)}$ and note $r^{2n+1} = 1$. Thus $\displaystyle\sum_{\ell=0}^{2n} e^{i(m-k)t_\ell} = \sum_{\ell=0}^{2n} r^\ell =$

$\dfrac{r^{2n+1}-1}{r-1} = 0$. From (32) $c_k = \dfrac{1}{2n+1} \displaystyle\sum_{\ell=0}^{2n} e^{-ikt_\ell} f(t_\ell) \quad k = -n, \cdots n.$

Further

$$\begin{cases} a_\ell = c_\ell + c_{-\ell} = \frac{1}{2n+1}\sum_{k=0}^{2n}(e^{-i\ell t_k} + e^{i\ell t_k})f(t_k) = \dfrac{2}{2n+1}\sum_{k=0}^{2n} f(t_k)\cos \ell t_k \\ \\ b_\ell = i(c_\ell - c_\ell) = \dfrac{2}{2n+1}\sum_{k=0}^{2n} f(t_k)\sin \ell t_k \end{cases}$$

□

Example 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 $p_2(t_\ell) = e^{\sin t_\ell + \cos t_\ell} \quad t_\ell = \dfrac{2\pi}{5}\ell$ 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 42

□

8 NUMERICAL DIFFERENTIATION

Recall that the derivative of a function $f(x)$ at a point x_0 is

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{33}$$

Computing the quotient numerically does not seem to be such a big problem. An efficient process by which we find an approximation to the limit is not clear. Actually, as we will present later on, even taking the quotient CAN be

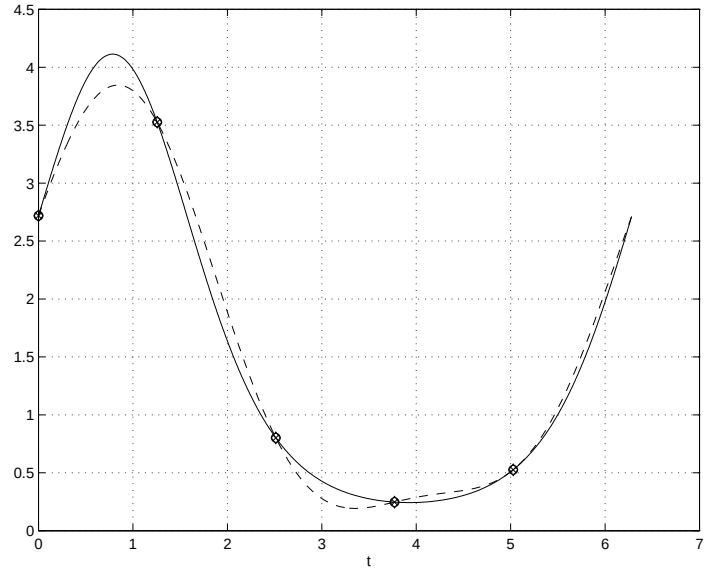

Figure 42: exact (solid), versus interpolated (dashed) versions of the function, using 5 interpolation points.

a problem on a computer: using finite precision we can wind up with serious loss of precision due to the subtraction and the division of quantities.

Here we will present one way to obtain derivatives, by divided differences. One can also obtain derivatives of functions by some other means: for L_2-periodic functions we might use spectral methods instead.

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.

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{1}{2}h^2 f''(\xi) \text{ with } \quad \xi \in (x_0, x_0 + h).$$

Therefore

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \frac{1}{2}hf''(\xi). \tag{34}$$

Comparison of (33) and (34) 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 (34) with $h > 0$ we obtain the "first order forward difference formulat 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

$$\frac{|hf''(\xi)|}{2} = \frac{|h|}{2\xi^2} \leq \frac{|h|}{2(1.8)^2} \text{ where } 1.8 < \xi < 1.8 + h$$

$$\nwarrow \text{ from } \; f''$$

h	f'	$\frac{\lvert h\rvert}{2\xi^2}$
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'	$\frac{\lvert h\rvert}{2\xi^2}$
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 diffentiating. 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".

□

More general derivative formulas:

A way to derive just about any derivative and to whatever truncation error you might want is via Lagrange polynomials (you can do this very efficiently using Newton divided differences, actually). We present here how this is done. 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.

Suppose $\{x_0, x_1, \cdots, x_n\}$ are $n+1$ distinct numbers, I is an interval containing such numbers, and $f \in C^{n+1}(I)$.

Using Lagrange polynomials ℓ, 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:

$$f'(x) = \sum_{k=0}^{n} f(x_k)\ell_k'(x) + \frac{d}{dx}\left[\frac{(x-x_0)\cdots(x-x_n)}{(n+1)!}\right] f^{(n+1)}(\xi(x)) + \frac{1}{(n+1)!}(x_0-x_0)\cdots(x-x_n)\frac{d}{dx}\left(f^{(n+1)}(\xi(x))\right)$$

What is the truncation error?

If $x = x_k$ then term involving $D_x[f^{n+1}(\xi)] = 0$. Therefore,

$$f'(x_k) = \sum_{j=0}^{n} f(x_j)\ell'_j(x_k) + \frac{f^{(n+1)}(\xi(x_k))}{(n+1)!}\Pi^n_{\substack{j=0\\j\neq k}}(x_k - x_j)$$

which is equivalent to the $n+1$-point formula, since

$$\frac{\partial}{\partial x}\prod_{j=0}^{n}(x - x_j) = \sum_{j=0}^{n}\prod_{\substack{i=0\\i\neq j}}^{n}(x - x_i), \qquad \text{if } x = x_k \Rightarrow \prod_{\substack{j=0\\j\neq k}}^{n}(x_k - x_j).$$

One might think that the higher the truncation error, the more accurate the answer. In general, using more points leads to greater accuracy. But more functional evaluations and possible 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.

Most common are forward and backward formulas. But other useful ones are the 3 and 5 point formulas:

$$\ell_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \quad \ell'_0(x) = \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}$$

$$\ell_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \quad \ell'_1(x) = \frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}$$

$$\ell'_2(x) = \frac{(2x - x_0 - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Therefore,

$$\begin{aligned} f'(x_j) = {} & f(x_0)\left[\frac{2xj - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] \\ & + f(x_1)\left[\frac{2xj - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] \\ & + f(x_2)\left[\frac{2xj - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right] + \frac{1}{6}f^{(3)}(\xi_j)\frac{d}{dx}\prod_{\substack{i=0\\i\neq j}}^{2}(x_j - x_i) \end{aligned} \tag{35}$$

for each $j = 0, 1, 2$ and $\xi_j = \xi(x_j)$

Use (35) on an evenly spaced grid. Let $x_1 = x_0 + h \quad x_2 = x_0 + 2h$, with $h \neq 0$

take $x_j = x_0 \quad , \quad x_1 = x_0 + h, \qquad x_2 = x_0 + 2h$

$$f'(x_0) = \frac{1}{h}\left[-\frac{3}{2}f(x_0) + 2f(x_1) - \frac{1}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_0)$$

Now take $x_j = x_i$

$$f'(x_1) = \frac{1}{h}\left[-\frac{1}{2}f(x_0) + \frac{1}{2}f(x_2)\right] - \frac{h^2}{3}f^{(3)}(\xi_1)$$

finally, when $x_j = x_2$,

$$f'(x_2) = \frac{1}{h}\left[\frac{1}{2}f(x_0) - 2f(x_1) + \frac{3}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_2)$$

Now, since $x_1 = x_0 + h, \quad x_2 = x_0 + 2h$ we can shift all of these so that we can compare them: we can translate to get

$$f'(x_0) = \frac{1}{2h}\left[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)\right] + \frac{h^2}{3}f^{(3)}(\xi_0)$$

$$f'(x_0) = \frac{1}{2h}\left[-f(x_0 - h) + f(x_0 + h)\right] - \frac{h^2}{6}f^{(3)}(\xi_1) \leftarrow \text{equiv by letting } h = -h$$

$$f'(x_0) = \frac{1}{2h}\left[f(x_0 - 2h) - 4f(x_0 - h) + 3f(x_0)\right] + \frac{h^2}{3}f^{(3)}(\xi_2)$$

But there are 2 formulas, really, since the last and first are equivalent by letting $h \to -h$.

3-point formulas : Recall that

$$f'(x_0) = \frac{1}{2h}[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)] + \frac{h^2}{3}f^{(3)}(\xi_0) \tag{36}$$

$$f'(x_0) = \frac{1}{2h}[f(x_0 + h) - f(x_0 - h)] - \frac{h^2}{6}f^{(3)}(\xi_1) \tag{37}$$

where

$$\xi_0 \text{ lies between } x_0 \text{ and } x_0 + 2h$$
$$\xi_1 \text{ lies between } (x_0 - h) \text{ and } (x_0 + h)$$

(37) error $\sim \frac{1}{2}$ (36) error, since data is examined on both sides of x_0.

In a similar way one can derive *5-point formulas*:

$$f'(x_0) = \frac{1}{12h}[f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)] + \frac{h^4}{30}f^5(\xi) \quad (38)$$

$$f'(x) = \frac{1}{12h}[-25f(x_0) + 48f(x_0 + h) - 36f(x_0 + 2h) + 16f(x_0 + 3h) - 3f(x_0 + 4h)] + \frac{h^4}{5}f^5(\xi) \quad (39)$$

$x_0 \leq \xi \leq x_0 + 4h$. Use $h < 0$ or $h > 0$ above to obtain left and right -going formulas

(36) and (39) are useful at end of the interval.

How to get higher-order derivatives? Can use same procedure as above.

But here we take a different more heuristic approach. Say you need to find the formula for the divided difference formula for $f''(x_0)$: expanding in Taylor series

Example Take f'' Second derivative of f, for example, found by expanding

$$f(x_0 + h) = f(x_0) + f'(x_0)h + f^{(''')}(x_0)\frac{h^2}{2} + f^{(3)}(x_0)\frac{h^3}{6} + f^{(4)}(\xi_{+h})\frac{h^4}{24} \text{ and}$$
$$f(x_0 - h) = f(x_0) - f'(x_0)h + f''(x_0)\frac{h^2}{2} - f^{(3)}(x_0)\frac{h^3}{6} + f^{(4)}(\xi_{-h})\frac{h^4}{24}.$$

Then, add and solve for f'':

$$f''(x_0) = \frac{1}{h^2}\Big[f(x_0 - h) - 2f(x_0) + f(x_0 + h)\Big] - \frac{h^2}{24}\Big[f^{(4)}(\xi_{-h}) + f^{(4)}(\xi_h)\Big]$$

If $f^{(4)}$ is continuous on $[x_0 - h, x_0 + h]$ the Intermediate Value Theorem yields

$$f''(x_0) = \frac{1}{h^2}[f(x_0 + h) - 2f(x_0) + f(x_0 - h)] - \frac{h^2}{12}f^{(4)}(\xi)$$
$$\text{for some } x_0 - h < \xi < x_0 + h.$$

□

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,

$$\begin{aligned} &\left| f'(x_0) - \frac{\tilde{f}(x_0+h) - \tilde{f}(x_0-h)}{2h} \right| \\ &\leq \frac{\varepsilon}{h} + \frac{h^2}{6}M. \end{aligned}$$

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 **??**).

In Figure 43 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 round off error of the computer becomes more important and the error starts to increase

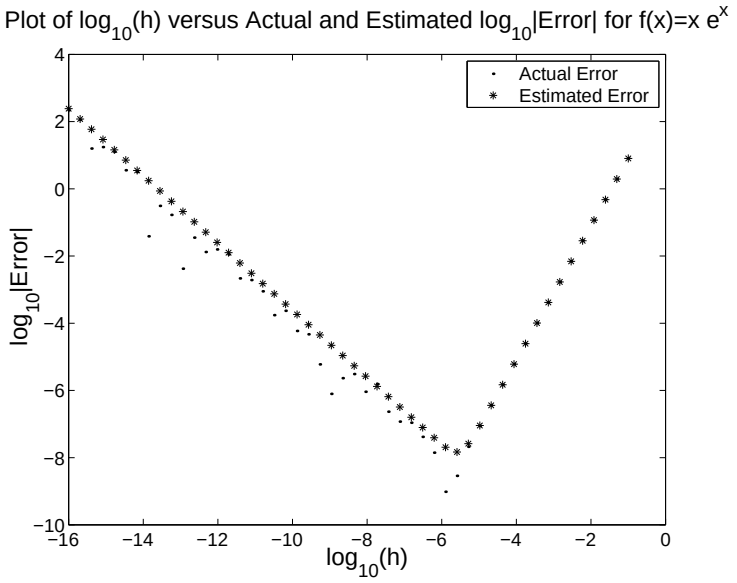

Figure 43: Error in the central difference approximation to the derivative $f'(x)$ as a function of h. The dots correspond to the error solely due to truncation error. Stars correspond to the the error from both truncation and roundoff.

with decreasing ϵ. 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 ϵ is $2 \times f(x_0) \times$ machine precision $= 2.4246 \times 10^{-14}$.

h	|Error|	Estimated Error
10^{-1}	8.124453	8.007729
10^{-2}	8.008886×10^{-2}	8.007729×10^{-2}
10^{-3}	8.007740×10^{-4}	8.007729×10^{-4}
10^{-4}	8.007935×10^{-6}	8.007971×10^{-6}
10^{-5}	8.172410×10^{-8}	8.250193×10^{-8}
10^{-6}	6.761866×10^{-9}	2.504722×10^{-8}
10^{-7}	2.348227×10^{-7}	2.424725×10^{-7}
10^{-8}	1.194055×10^{-6}	2.424645×10^{-6}
10^{-9}	1.763533×10^{-5}	2.424645×10^{-5}
10^{-10}	7.233669×10^{-6}	2.424645×10^{-4}
10^{-11}	1.704020×10^{-4}	2.424645×10^{-3}
10^{-12}	1.757870×10^{-2}	2.424645×10^{-2}
10^{-13}	1.493988×10^{-1}	2.424645×10^{-1}
10^{-14}	2.230919	2.424645
10^{-15}	1.875648	24.24645
10^{-16}	218.3926	242.4645

□

9 NUMERICAL INTEGRATION

The general methodology in approximating Riemann integrals is to replace the integral by a finite sum: For example,

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} c_i f(x_i)$$

This procedure is called *numerical Quadrature*. This 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 freedom we have to devise methods by which to do this comes in choosing where to evaluate the function, and in choosing what the coefficients c_i should be. One very natural way to do this is with Lagrange polynomials:

Assume $\{x_i\}_{i=0}^n$ are $n+1$ distinct points. Then

$$P_n(x) = \sum_{i=0}^{n} f(x_i) l_i(x). \tag{40}$$

Here $l_i(x)$ are just the cardinal functions. Replacing,

$$\begin{aligned}
\int_a^b f(x)\,dx &= \int_a^b \sum_{i=0}^{n} f(x_i) l_i(x)\,dx + \int_a^b \underbrace{\prod_{i=0}^{n}(x-x_i)\frac{f^{(n+1)}(\xi(x))}{(n+1)!}\,dx}_{\text{error in approximation of } f(x)} \\
&\equiv \underbrace{\sum_{i=0}^{n} c_i f(x_i)}_{\text{quadrature formula}} + \underbrace{\frac{1}{(n+1)!}\int_a^b \prod_{i=0}^{n}(x-x_i) f^{(n+1)}(\xi(x))dx}_{\text{error E(f)}}
\end{aligned}$$

where $\xi(x) \in [a,b]$ for each x. This then determines what the values of c_i are, where c_i are known as the *quadrature weights*:

$$c_i = \int_a^b l_i(x)\,dx, \text{ for each } i = 0, 1, \ldots, n.$$

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 $l_i(x)$. Let $x = a + hu$, then $u = (x-a)/h$. Hence,

$$\begin{aligned}
c_i &= \int_a^b \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}\,dx \\
&= h\int_0^n \frac{\prod_{j\neq i}(u-j)}{\prod_{j\neq i}(i-j)}\,du \\
&\equiv h\gamma_i
\end{aligned}$$

Notice that the value of γ_i is a number which is independent of a and b, dependent only on n.

The formulas for 1^{st} and 2^{nd} Lagrange polynomials at equally-spaced intervals generate the Trapezoidal Rule (see 9.1) and Simpson's Rule (see 9.2).

9.1 Trapezoidal Rule:

Here we consider the case $n = 1$ in (40), i.e. linear approximation. In this case $h = b - a$ and $\gamma_0 = \int_0^1 \frac{u-1}{0-1}\,du = \frac{1}{2}$, 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

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left(f(a) + f(b)\right).$$

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

$$\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx$$

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 $f(x)g(x) \leq (\bar{f})g(x) - \epsilon$. Integrating from a to b and splitting the interval into pieces we get

$$\int_a^b f(x)g(x)\,dx \;=\; \int_a^{d-\delta} f(x)g(x)\,dx + \int_{d-\delta}^{d+\delta} f(x)g(x)\,dx + \int_{d+\delta}^{b} f(x)(g(x)\,dx$$

$$\leq \bar{f}\left(\int_a^{d-\delta} g(x)\,dx + \int_{d-\delta}^{d+\delta} g(x)\,dx + \int_{d+\delta}^{b} g(x)\,dx\right) - 2\delta\epsilon$$
$$= \int_a^b g(x)\,dx - 2\delta\epsilon < \int_a^b g(x)\,dx.$$

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

$$h^3 \int_0^1 f''(\xi(a+hu))\frac{u(u-1)}{2}\,du = h^3 f''(\xi)\int_0^1 \frac{u(u-1)}{2}\,du$$
$$= h^3 f''(\xi)\Big[\frac{u^3}{6} - \frac{u^2}{4}\Big|_0^1 = -\frac{h^3}{12}f''(\xi)$$

Putting it all together we have

$$\int_a^b f(x)dx = \frac{h}{2}\left(f(a)+f(b)\right) - h^3\frac{f''(\xi)}{12}$$

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 44

Exercise An important fact about the trapezoidal rule is obtained by considering the approximation of the integral of $f(x)$ bounded and continuous, with $f(x+T) = f(x)$ for all x, i.e. periodic with period T. □

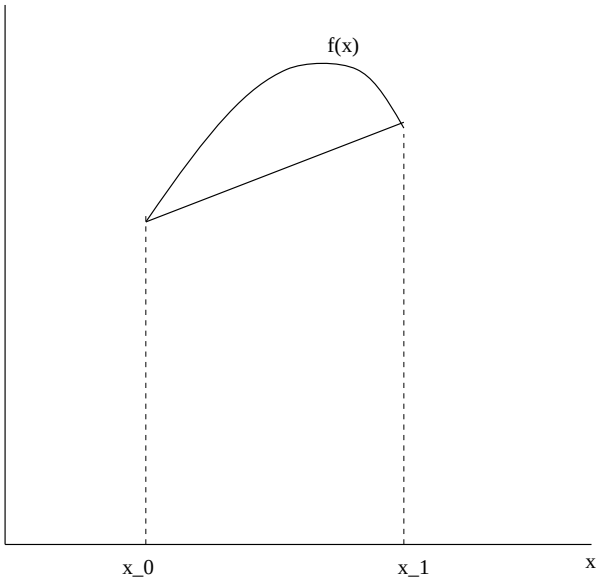

Figure 44: The trapezoidal rule approximates the area under $f(x)$ by a trapezoid.

9.2 Simpson's Rule:

Unlike the trapezoidal case which uses linear polynomials, the Simpson Rule is what one obtains from using second-order Lagrange polynomials, i.e. quadratic functions. Schematically, the approximation of the area under the function $f(x)$ over a finite interval is is shown in Figure 45

Let $h = \dfrac{b-a}{2}, \quad x_0 = a, \quad x_1 = a+h, \quad x_2 = b$, then, integrating the Lagrage polynomial we obtain

$$\int_a^b f(x)dx =$$
$$\int_{x_0}^{x_2} \left[\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) \right.$$
$$\left. + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2) \right] dx +$$
$$\int_{x_0}^{x_2} \frac{1}{6}(x-x_0)(x-x_1)(x-x_2) f^{(3)}(\xi(x))dx$$

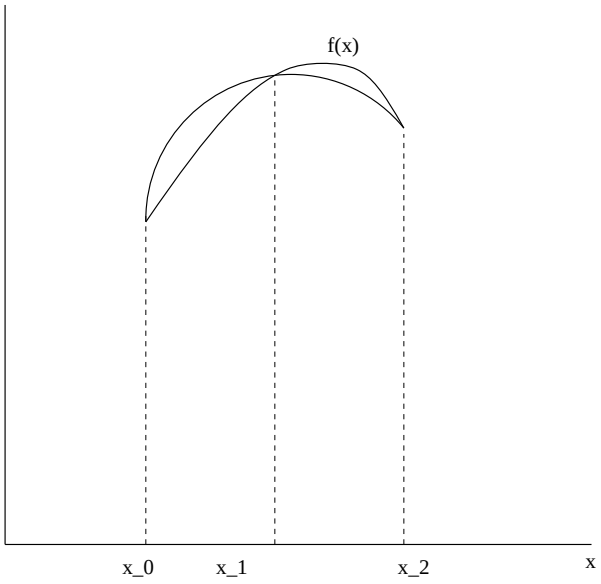

Figure 45: The Simpson rule approximates the area under $f(x)$ by a box bounded above by a parabolic function.

Exercise Find the error applying the same technique used in the trapezoidal case. You should get $O(h^4)$ error involving $f^{(3)}$

□

There is nothing special about how we went about constructing Simpson's. We can get a better approximation by using Taylor series as follows:

$$f(x) = f(x_1)+f'(x_1)(x-x_1)+\frac{f''(x_1)}{2}+(x-x_1)^2+\frac{f'''(x_1)}{6}(x-x_1)^3+\frac{f^{(iv)}(\xi(x))}{24}(x-x_1)^4$$

and integrating both sides of this equation it follows that

$$\begin{aligned}(41)\qquad \int_{x_0}^{x_2} f(x)dx &= f(x_1)(x_2-x_0)+ \\ &\left[\frac{f'(x_1)}{2}(x-x_1)^2+\right. \\ &\left.\frac{f''(x_1)}{6}(x-x_1)^3+\frac{f'''(x_1)}{24}(x-x_1)^4\right]_{x_0}^{x_2}+ \\ &\frac{1}{24}\int_{x_0}^{x_2} f^{(iv)}(\xi(x))(x-x_1)^4dx.\end{aligned}$$

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

$$\frac{1}{24}\int_{x_0}^{x_2} f^{(iv)}(\xi(x))(x-x_1)^4 dx = \frac{f^{(iv)}(\xi_1)}{24}\int_{x_0}^{x_2}(x-x_1)^4 dx = \frac{f^{(iv)}(\xi_1)}{120}(x-x_1)^5\Big|_{x_0}^{x_2}$$

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 (41) as

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{12}\left[\frac{1}{3}f^{(iv)}(\xi_2) - \frac{1}{5}f^{(iv)}(\xi_1),\right].$$

(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

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.

Remark: Both Trapezoidal and Simpson Rules are examples of what are known as *Newton-Cotes Formulas*, since they 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 larger than 8 can lead to unusual results, since the weights or coefficients are negative, beyond this point.

Two simple ways of improving the accuracy of the quadrature rules just discussed is to use different sized panels (see9.4), or using unequally spaced points. For example, we could use Gaussian Quadrature (see 9.3), which we cover briefly next.

9.3 Gaussian Quadrature

In this strategy we want to choose the points in x 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 $c_1, c_2 \cdots c_n$.

$$|| \int_a^b f(x)dx - \sum_{i=1}^{n} c_i f(x_i)|| < \varepsilon,$$

where $\varepsilon > 0$ is the error we are willing to live with, in some appropriate norm.

The nodes are $x_i \in [a, b]$ (are not necessarily equally-spaced). So we need to fix the value of $2n$ quantities: the nodes x_i and the weights c_i.

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. $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$.

Since $\int (a_0 + a_1x + a_2x^2 + a_3x^3)dx = a_0 \int dx + a_1 \int xdx + a_2 \int x^2dx + a_3 \int x^3dx$, We can reduce the problem of choosing the c_i x_i to is equivalent to choosing them when $f(x) = 1,\ x,\ x^2,\ x^3$ (a basis of $P^{(4)}$. These are the necessary conditions:

$$c_1 + c_2 = \int_{-1}^{1} 1dx = 2 \qquad c_1x_1 + c_2x_2 = \int_{-1}^{1} xdx = 0$$
$$c_1x_1^2 + c_2x_2^2 = \int_{-1}^{1} x^2dx = \frac{2}{3} \quad c_1x_1^3 + c_2x_2^3 = \int_{-1}^{1} x^3dx = 0$$

Algebra shows that

$$c_1 = 1, c_2 = 1, x_1, = \frac{\sqrt{3}}{3}, x_2 = \frac{\sqrt{3}}{3}$$
$$\therefore \int_{-1}^{1} f(x)dx \approx f\left(\frac{-\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$$

which gives exact results when $f \in \mathcal{P}^3$ □

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 $[0, 2\pi]$ with periodic functions, use Chebyshev polynomials on $[-1, 1]$, etc.

For example, suppose we use Legendre Polynomials. Properties of Legendre Polynomials

1. For each n, P_n is polynomial of degree n.
2. $\int_{-1}^{1} P(x)P_n(x)dx = 0$ whenever $P(x)$ is polynomial of degree less than n.

The first few Legendre polynomials are:

$$P_0(x) = 1 \quad P_1(x) = x \quad P_2 = x^2 - \frac{1}{3} \quad P_3 = x^3 - \frac{3}{5}x, \quad P_4 = x^4 - \frac{6}{7}x^2 + \frac{3}{35}$$

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 $i = 1, 2, \cdots n$

$$c_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} dx$$

If P is a polynomial of degree less than $2n$ then $\displaystyle\int_{-1}^{1} P(x)dx = \sum_{j=1}^{n} c_j P(x_j)$

In this Gaussian Quadrature, the error is

$$\begin{aligned} E_n(f) &= \frac{f^{(2n)}(\xi)}{(2n)!} \int_{-1}^{1} P_n^2(x)dx \\ &= \frac{2}{(2n+1)!} \left[\frac{2^n (h!)^2}{(2n)!} \right]^2 f^{(2n)}(\xi) \quad -1 < \xi < 1 \end{aligned}$$

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,

$$\int_{-1}^{1} R(x)dx = \int_{-1}^{1} \left[\sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} R(x_i) \right] dx \tag{42}$$

$$= \sum_{i=1}^{n} \left[\int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_j - x_j} dx \right] R(x_i) = \sum_{i=1}^{n} c_i R(x_i)$$

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

$$\int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} c_i R(x_i)$$

so

$$\begin{aligned}\int_{-1}^{1} P(x)dx &= \int_{-1}^{1} \left[Q(x)P_n(x) + R(x)\right] dx \\ &= \int_{-1}^{1} R(x)dx = \sum_{i=1}^{n} c_i R(x_i) = \sum_{i=1}^{n} c_i P(x_i)\end{aligned}$$

□

To find the values of the roots of P_n consult 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

$$t = \frac{1}{b-a}(2x - a - b) \text{ then } [a, b] \to [-1, 1]$$

$$\text{and } \int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{(b-a)t + b + a}{2}\right) \frac{(b-a)}{2} dt$$

□

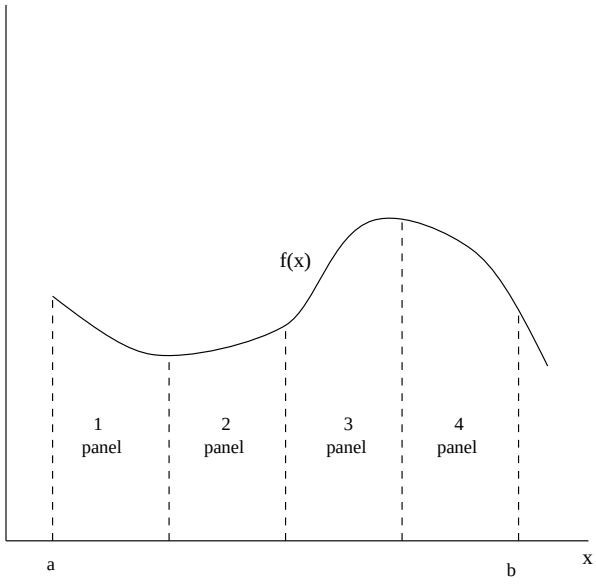

Figure 46: Composite integration methods split the total integration domain into smaller parts.

9.4 Composite Numerical Integration

Newton-Cotes formulas are unsuitable for large integration intervals. Instead we 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 46.

$$\int_a^b f(x)dx = \int_{a_0=a}^{a_1} f(x)dx + \int_{a_1}^{a_2} f(x)dx + \int_{a_2}^{a_3} f(x)dx + \cdots + \int_{a_{n-1}}^{a_n=b} f(x)dx$$

with $a_i = a + i(b-a)/m$ 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 $x_j = a + jh$ with $h = (b-a)/n$ and $n = 2m$. The resulting formula is:

$$\int_a^b f(x)\,dx = \frac{h}{6}(\quad f(x_0) \quad +4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots$$
$$+4f(x_{n-1}) + f(x_n)) - \frac{h^5}{90}\sum_{i=0}^{n/2-1} f^{(iv)}(\xi_i)$$

Using the weighted mean value theorem it is straightforward to replace the error term with a single term

$$\int_a^b f(x)\,dx = \frac{h}{6}(\quad f(x_0) \quad +4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots$$
$$+4f(x_{n-1}) + f(x_n)) - \frac{nh^5}{2\cdot 90} f^{(iv)}(\xi)$$

and since $h = (b-a)/n$ this can be written as

$$\int_a^b f(x)\,dx = \frac{h}{6}(\quad f(x_0) \quad +4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots$$
$$+4f(x_{n-1}) + f(x_n)) - \frac{h^4(b-a)}{180} f^{(iv)}(\xi) \tag{43}$$

Notice that in going from Simpson's to composite Simpson's rule, the error went from being $O(h^5)$ to $O(h^4)$ Also for this rule n must be even.

A similar, but simpler argument (in this case $m = n$, and n can be any positive number)

using the trapezoid rule yields

$$\int_a^b f(x)\,dx = \frac{h}{2}(\quad f(x_0) \quad +2f(x_1) + 2f(x_2) + 2f(x_3) + 2f(x_4) + \cdots$$
$$+2f(x_{n-1}) + f(x_n)) - \frac{h^2(b-a)}{12} f''(\xi) \tag{44}$$

Example: Lets look at using the composite method for evaluating the integral

$$\int_0^5 \mathrm{e}^x dx.$$

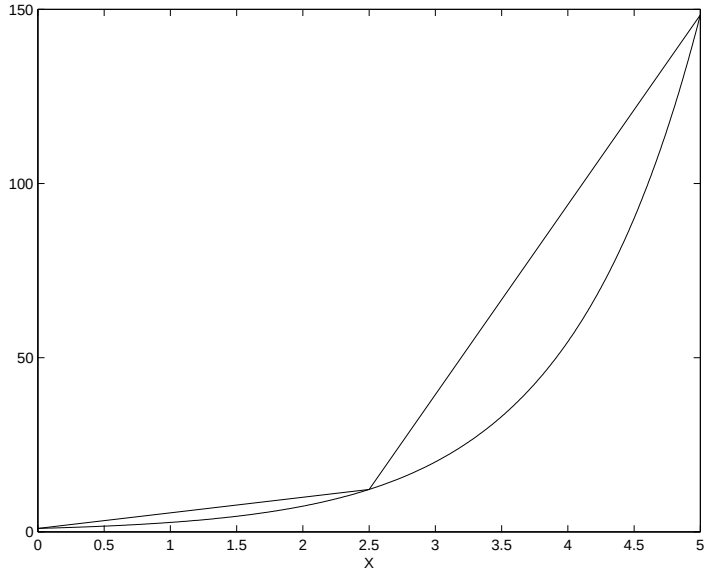

Figure 47: Plot of the function $f(x) = \mathrm{e}^x$ over the interval of integration along with the trapezoidal approximation for $n = 2$

We know that the actual value of this integral is $\mathrm{e}^5 - \mathrm{e}^0 = 147.4132$. 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 47 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 48. 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.

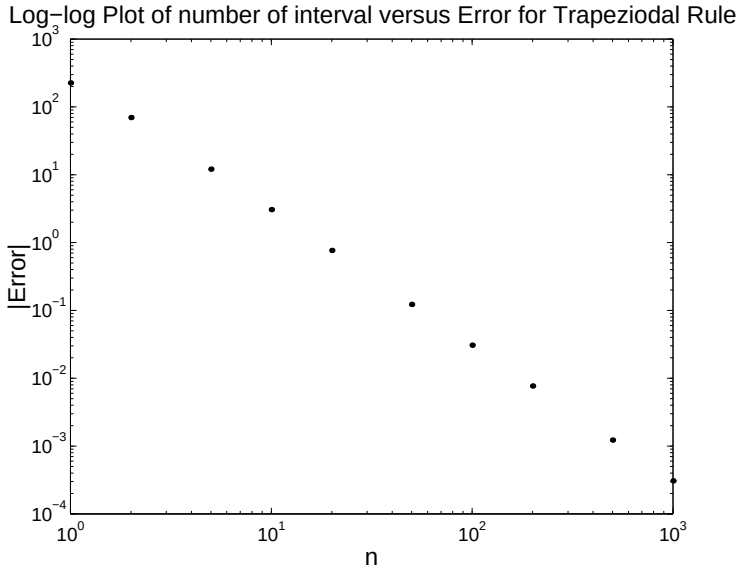

Figure 48: Log-log plot showing the absolute error of the Trapezoidal rule numerical integration of $\int_0^5 \mathrm{e}^x dx$ for different numbers of intevals n.

n	Approximate Integral	Absolute Error
1	373.5329	226.12
2	217.2227	69.810
5	159.4976	12.084
10	150.4715	3.0584
20	148.1801	0.76698
50	147.5360	0.12282
100	147.4439	3.0710×10^{-02}
200	147.4208	7.6777×10^{-03}
500	147.4144	1.2284×10^{-03}
1000	147.4135	3.0711×10^{-04}

We can also use Simpson's rule to estimate the integral. Simpson's Rule requires an even number of intervals. Figure 49 shows $f(x) = \mathrm{e}^x$ 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 below shows the absolute error for

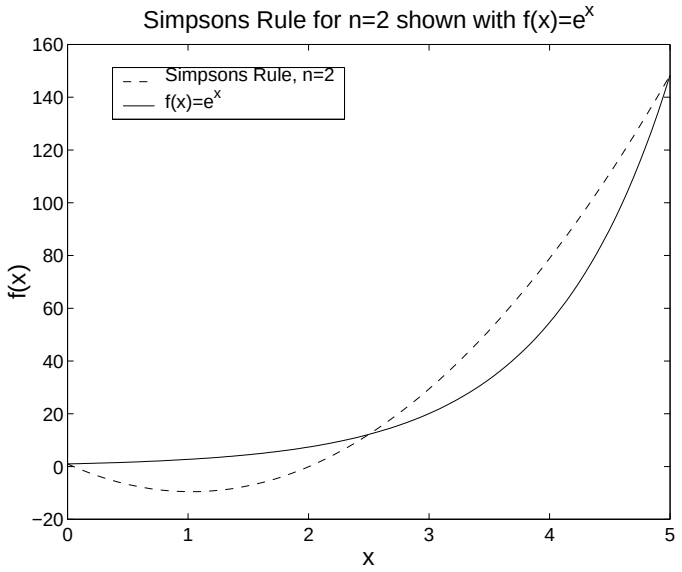

Figure 49: Plot of the function $f(x) = \mathrm{e}^x$ over the interval of integration along with the Simpson's Rule approximation for $n = 2$

n going from two to two thousand. Figure 50 also shows this graphically on a log-log plot. Again we have a power law relationship between n and —$Error$— but in this case it is

$$Error(n) \approx Cn^4$$

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 5.1187×10^{-10} compared with an error of 3.0711×10^{-04} for the trapezoidal rule.

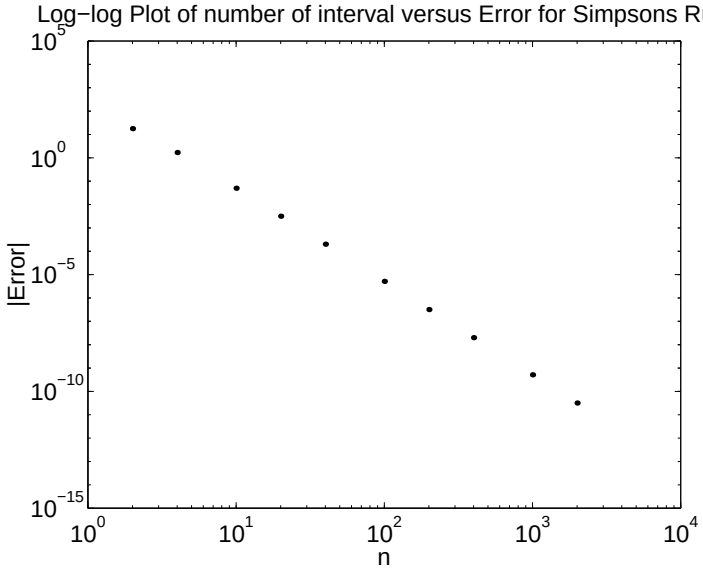

Figure 50: Log-log plot showing the absolute error of Simpson's rule numerical integration of $\int_0^5 \mathrm{e}^x dx$ for different numbers of intervals n.

n	Approximate integral	Absolute Error
2	165.1193	17.706
4	149.0933	1.6801
10	147.4629	4.9701×10^{-02}
20	147.4163	3.1754×10^{-03}
40	147.4134	1.9957×10^{-04}
100	147.4132	5.1170×10^{-06}
200	147.4132	3.1988×10^{-07}
400	147.4132	1.9994×10^{-08}
1000	147.4132	5.1187×10^{-10}
2000	147.4132	3.1974×10^{-11}

9.5 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.

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).

9.6 Monte Carlo Method for Integrals

We present here the simplest application of Monte Carlo techniques. We will use Monte Carlo to approximate the integral

$$\int_a^b f'(x')\,dx'.$$

Without loss of generality, we will be further assuming that the function $f'(x')$ can be transformed to $0 \le f(x) \le 1$, and that the limits can be mapped to the interval 0 and 1, respectively, so that the task is now converted to approximating the integral

$$I \equiv \int_0^1 f(x)\,dx.$$

Suppose we choose N pairs of points (x_i, y_i) with uniform distribution (see Figure 51) and define

$$z_i = \left[\begin{array}{ll} 0 & y_i > f(x_i) \\ 1 & y_i \le f(x_i). \end{array} \right].$$

See Figure 52. Then, if $n = \sum_i z_i$, we have $n/N \approx I$, or more precisely,

$$I = \frac{n}{N} + O(1/\sqrt{N}).$$

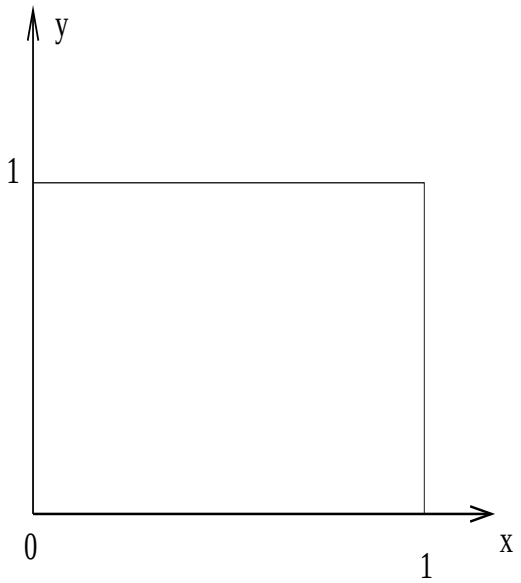

Figure 51: Domain of (x, y) pairs.

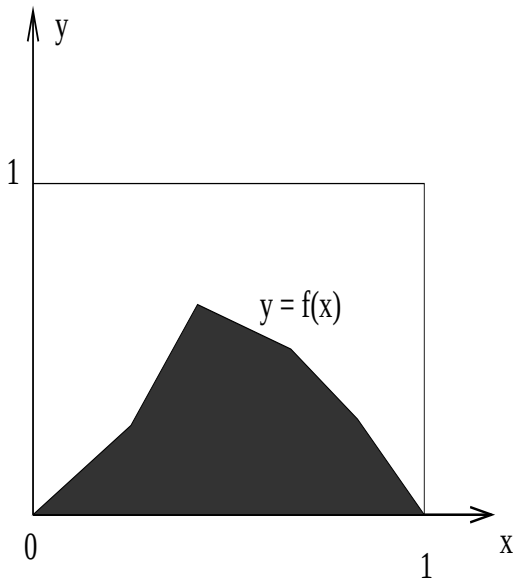

Figure 52: The value of z_i is 0 for $y_i > f(x_i)$, and 1 if in the shaded region.

Can also consider I as the mean value of the probability density function $f(\xi)$ with ξ 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 the integral in $\mathbf{R}^n$ space of f as

$$(45)\quad \int_V f(y_1, y_2, \ldots, y_n)dy_1 dy_2 \ldots dy_n \approx V < f > \pm\sqrt{\frac{< f^2 > - < f >^2}{N}},$$

where

$$\begin{aligned} < f > &\equiv \frac{1}{N}\sum_{i=1}^{N} f(y_i) \\ < f^2 > &\equiv \frac{1}{N}\sum_{i=1}^{N} f^2(y_i) \end{aligned}$$

so we recognize the last term on the right hand side of (45) as the volume V times the standard deviation.

This is not a rigorous bound and it assumes that the error is Gaussian distributed. In fact, it also needs to be pointed out that the the approximation also depends on the quality of the random number generator and the actual creation of good and fast random number generators is a matter of current research.

The important thing about (45) is that it says that the integral converges, but very slowly: $O(1/\sqrt{N})$, 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.

10 RICHARDSON EXTRAPOLATION

Used to generate high accuracy by using low-order formulas.

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.

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. In fact we know what the explicit expression is for the first few terms in the truncation error.

$$M = N(h) + K_1 h + \underbrace{K_2 h^2 + K_3 h^3}\cdots \qquad (46)$$

K's are known constants.

truncation error[5] 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 (46) when $\frac{h}{2}$

$$M = N\left(\frac{h}{2}\right) + K_1\frac{h}{2} + K_2\frac{h^2}{4} + K_3\frac{h^3}{8} + \cdots \qquad (47)$$

Multiply (47) by 2 and subtract (46) to obtain

$$M = \left[N\left(\frac{h}{2}\right) + \left(N\left(\frac{h}{2}\right) - N(h)\right)\right] + K_2\left(\frac{h^2}{2} - h^2\right) + K_3\left(\frac{h^3}{4} - h^3\right) + \cdots$$

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)

$$M = N_2(h) - \frac{K_2}{2}h^2 - \frac{3K_3}{4}h^3 + \cdots \qquad (48)$$

[5] φ in Kincaid and Chenney.

Now, replace h by $\dfrac{h}{2}$

$$M = N_2\left(\frac{h}{2}\right) - \frac{K_2}{8}h^2 - \frac{3K_3}{32}h^3\cdots \tag{49}$$

Subtract 4 times (49) minus (48):

$$3M = 4N_2\left(\frac{h}{2}\right) - N_2(h) + \frac{3K_3}{4}\left(-\frac{h^3}{2} + h^3\right) + \cdots$$

$$\text{So } M = \left[N_2\left(\frac{h}{2}\right) + \frac{N_2(h/2) - N_2(h)}{3}\right] + \frac{K_3}{8}h^3\cdots$$

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 $\mathcal{O}(h^5)$ is

$$N_5(h) = N_4\left(\frac{h}{2}\right) + \frac{N_4(h/2) - N_4(h)}{15} \text{ and so on .}$$

$$\text{Thus, if } M = N(h) + \sum_{j=1}^{m-1} K_j h^j + O(h^m)$$

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 $\displaystyle\sum_{j=1}^{m-1} K_j h_{\alpha j} + O(h_m^{\alpha})\, \alpha_1 < \alpha_2 \cdots < \alpha_m.$

note that the $K's$ are known.

Example

$$f'(x_0) = \frac{1}{2h}\left[f(x_0+h) - f(x_0-h)\right] - \frac{h^2}{6}f'''(x_0) - \frac{h^4}{120}f^v(x_0)\cdots$$

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 $O(h^{2j})$

$$N_j(h) = N_{j-1}\left(\frac{h}{2}\right) + \frac{N_{j-1}(h/2) - N_{j-1}(h)}{4^{j-1}-1},$$

↖ 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.

□

Discussion: 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. Furthermore, it'll increase the possibility of instabilities in numerical differentiation.

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 $\left\{ \mathbf{x}^{(k)} \right\}_{k=1}^{\infty}$ to some $\mathbf{x}$.

definition: Euclidean length $|\mathbf{x}| \equiv \sqrt{(x_1)^2 + (x_2)^2 \cdots (x_n)^2}$, for $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Let $\mathbb{R}^n \equiv$ set all n-dimensional vectors with real components

$$R^n = \left\{ \mathbf{x} | \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad , x_1, x_2, \ldots, x_n \text{ real} \right\}$$

A norm in $\mathbb{R}^n$ is a real-valued function $|| \cdot ||$ defined on $\mathbb{R}^n$ satisfying

$||\mathbf{x}|| \geq 0$, and $||\mathbf{x}|| = \mathbf{0}$ if and only if $\mathbf{x} = \mathbf{0}$
$||\alpha \mathbf{x}|| = |\alpha| \, ||\mathbf{x}|| \quad \forall \, \alpha$ scalars and $\mathbf{x}$ vectors $\in \mathbb{R}^n$.
$||\mathbf{x} + \mathbf{y}|| \leq ||x|| + ||y|| \, \forall \mathbf{x}$ and $\mathbf{y} \in \mathbb{R}^n$
Note: A useful variation $||\mathbf{x} - \mathbf{y}|| \geq ||\mathbf{x}|| - ||\mathbf{y}||$
Three $l_p -$ norms for $\mathbb{R}^n, p = 1, 2, \infty,$ are :

$$||x||_1 = |x_1| + |x_2| + \ldots |x_n|$$
$$||x||_2 = \sqrt{(x_1)^2 + (x_2)^2 \cdots (x_n)^2}$$
$$||x||_\infty = \max \{|x_1|, |x_2|, \cdots |x_n|\}$$

□

Example Take

$\mathbf{x} = [1 - 2 - 2]^T$ then $\quad ||\mathbf{x}||_1 = 5 \quad ||\mathbf{x}||_2 = 3$ □

and $\quad ||\mathbf{x}||_\infty = 2$

Remark:

With these norms, the concept of relative and absolute error are thus defined

$$||e||_p^A = ||\mathbf{x} - \widehat{\mathbf{x}}||_p \quad \text{and} \quad ||e||_p^R = \frac{||\mathbf{x} - \widehat{\mathbf{x}}||_\mathbf{p}}{||\mathbf{x}||_p}$$

$p = 1, 2, \infty.$

Example Consider subspace $\{\mathbf{x} : \mathbf{x} \in \mathbb{R}^2, ||\mathbf{x}|| \leq 1\}$

The l_2-norm $||\mathbf{x}||_2 \leq 1$ 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_∞-norm $||x||_\infty \leq 1$ defines a "unit ball" that in the 2-d plane is a square. □

In general: $||\mathbf{x}||_1 \geq ||\mathbf{x}||_2 \geq ||\mathbf{x}||_\infty$

Also $||\mathbf{x}||_1 \leq \sqrt{n}||\mathbf{x}||_2 \quad , \quad ||\mathbf{x}||_2 \leq \sqrt{n}||\mathbf{x}||_\infty$ and $||\mathbf{x}||, \leq n||\mathbf{x}||_\infty$

11.1.1 Matrix Norms (Undergraduate Students)

let M_n denote $(n \times n)$ matrices, $\mathbf{0}$ 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 $B \in M_n$

$$
\begin{aligned}
&||A|| \geq 0 \text{ and } ||A|| = 0 \text{ if and only if } A = \mathbf{0} \\
&||\alpha A|| = |\alpha|\,||A|| \text{for any scalar } \alpha \\
&||A + B|| \leq ||A|| + ||B|| \\
&||AB|| \leq ||A||\,||B||
\end{aligned}
$$

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

$$
\begin{aligned}
||A||_1 &= \max_{1 \leq j \leq n} \left[\sum_{i=1}^{n} |a_{ij}| \right] \text{ max column sum} \\
||A||_a &= \max_{1 \leq i \leq n} \left[\sum_{j=1}^{n} |a_{ij}| \right] \text{ max row sum} \\
||A||_E &\equiv \sqrt{\sum_i \sum_j (a_{ij})^2} \text{ Frobenius Norm}
\end{aligned}
$$

these are compatible, since

$$||A\mathbf{x}||_1 \leq ||A||_1||\mathbf{x}||_1 \quad , \quad ||Ax||_\infty \leq ||A||_\infty||\mathbf{x}||_\infty \quad , \quad ||A\mathbf{x}||_2 \leq ||A||_E||\mathbf{x}||_2$$

Example Suppose $A\mathbf{x} = b$ $\quad \mathbf{x}_c$ is approximation of $\mathbf{x}_t$ the exact solution.

$$\text{let } \mathbf{r} = A\mathbf{x}_c - \mathbf{b} \quad \text{be residual vector}$$

$$(50) \qquad \begin{cases} \text{then } \mathbf{r} = A\mathbf{x}_c - \mathbf{b} = A\mathbf{x}_c - A\mathbf{x}_t \text{ or} \\ \mathbf{x}_c - \mathbf{x}_t = A^{-1}\mathbf{r} \\ ||x_{\mathbf{c}} - x_t||_1 = ||A^{-1}\mathbf{r}||_1 \leq ||A^{-1}||, ||r||_1 \end{cases}$$

□

11.1.2 Matrix Norms (Graduate Students)

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 $V \to \mathbb{R}$ which satisfies the following "norm axioms."

(a) $||x|| \geq 0$, where $||x|| = 0$ when $x = 0$.

(b) $||\alpha x|| = |\alpha|\,||x|| \quad ,$ here $\alpha \in \mathbb{R}$

(c) $||x + y|| \leq ||x|| + ||y||$ "Triangle Inequality"

for $x \in V$ 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 $x \in V$

$$\begin{aligned} ||x||_2 &\equiv \sqrt{x_1^2 + x_2^2, \cdots, x_n^2} \\ ||x||_1 &\equiv |x_1| + |x_2| +, \cdots, |x_n| \\ ||x||_p &\equiv [|x_1|^p + |x_2|^p + \cdots + |x_n|^p]^{\frac{1}{p}} \\ ||x||_{\max} \equiv ||x||_\infty &= \max_{1 \leq j \leq n} [|x_j|] \end{aligned}$$

The last one comes from $||x||_p \to ||x||_{\max}$ as $p \to \infty$.

Linear Transformations:

The general linear transformation from $\mathbb{R}^n \to \mathbb{R}^m$ is of the form

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \to \begin{pmatrix} A_{11}x_1 & +A_{12}x_2 & +A_{13}x_3 & +\cdots & +A_{in}x_n \\ A_{21}x_1 & +\cdots & & & \\ \vdots & & & & \\ A_{m1}x_1 & +A_{m2}x_2 & +A_{m3}x_3 & +\cdots & A_{mn}x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

or

$$Ax = y.$$

A is an $m \times n$ matrix with entries $A_{i,j}$, $(i = 1, 2 \ldots m$ and $j = 1, 2, \cdots, n)$.

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.

$$tr(A) = \sum_{j=1}^{n} A_{jj} \quad , \text{ here } A \in \mathbb{R}^{n\times n}.$$

Can define norms on $\mathbb{R}^{m\times n}$ just as we did in $\mathbb{R}^n$. For example

$||A||_F = \sqrt{\sum_{j=1}^{n}\sum_{k=1}^{m} A_{ij}^2}$ "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 take the largest value.

It turns out that the quotient is always bounded $\Rightarrow$ 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 infinitely-dimensional spaces (e.g. Banach), the associated operator norm is

$$\sup_{x\neq 0} \frac{||Ax||_\beta}{||x||_\alpha}, \text{ even though supremum is not necessarily attained.}$$

Where x an element of the Banach space, A is an operator acting in the Banach space. □

Need to check that norm axioms apply. The first two follow from the definition. Check triangle inequality:

Suppose $A, B \in \mathbb{R}^{m\times n}$

$$||A+B||_{\alpha,\beta} \equiv \max_{x\neq 0} \frac{||(A+B)_x||_\beta}{||x||_\alpha} = \max_{x\neq 0} \frac{||Ax+Bx||_\beta}{||x||_\alpha}$$

$$\leq \max_{x\neq 0} \frac{||Ax||_\beta + ||Bx||_\beta}{||x||_\alpha} \leq \max_{x\neq 0} \frac{||Ax||_\beta}{||x||}_\alpha + \max_{x\neq 0} \frac{||Bx||_\beta}{||x||_\alpha} \therefore$$

$$||A+B||_{\alpha,\beta} \leq ||A||_{\alpha,\beta} + ||B||_{\alpha,\beta}$$

□

Two other vital properties which prove useful:

$$||Ax||_\beta \leq ||A||_{\alpha,\beta}||x||_\alpha$$
$$||AB||_{\alpha,\gamma} \leq ||A||_{\alpha,\beta}||B||_{\beta,\gamma}$$
$$\text{if } B \in \mathbb{R}^{n\times p} \quad , \quad ||\cdot||_\gamma \text{ on } \mathbb{R}^P.$$

□

So the vector norms $||\cdot||_p$, for $u \in \mathbb{R}^n$:

$$||u||_1 = \sum_{i=1}^{n} |u_i|$$

$$||u||_2 = \left(\sum_{i=1}^{n} |u_i|^2 \right)^{\frac{1}{2}}$$

$$||u||_p = \left(\sum_{i=1}^{n} ||u_i|^p \right)^{\frac{1}{p}} \quad , \quad 1 \leq p < \infty$$

$$||u||_\infty = \max_{i=1,n} \{|u_i|\}$$

Remark: To show that $|| \cdot ||_p \to || \cdot ||_\infty$ as $p \to \infty$ use L'Hopital's Rule.

Associated Matrix Norms of $A \in \mathbb{R}^{n \times n}$

$$||A||_1 = \max_{j=1,n} \left\{ \sum_{i=1}^{n} |a_{ij}| \right\}$$

$$||A||_2 = \max \left\{ \sqrt{\lambda} > 0, \text{ where } \lambda \in \sigma(A^T A) \right\}$$

where $\sigma(\cdot)$ is the eigenvalue spectrum with components λ.

$$||A||_\infty = \max_{i=1,n} \left\{ \sum_{j=1}^{n} |a_{ij}| \right\}$$

Remark: To prove $||A||_2$ is given in terms of spectrum, square A and use Lagrange multipliers and the definition of the associated norm.

□

Let $A \in \mathbb{R}^{n \times m} \quad , \quad b \in \mathbb{R}^n$

When does $Au = b$ have a unique solution $u \in \mathbb{R}^n$?

Theorem: The following statements are equivalent:

$$\exists\, !\, u \text{ such that } Au = b \iff \det(A) \neq 0$$

$$\iff Au = 0 \text{ has solution } u = 0$$

$$\iff A \text{ is invertible}$$

if $|| \cdot ||$ is a vector norm in $\mathbb{R}^n$ then

$$\iff \exists \alpha > 0 \text{ such that } ||Au|| \geq \alpha ||u|| \quad \forall u$$

moreover $||A^{-1}|| \leq \dfrac{1}{\alpha}$ where

$|| \cdot ||$ is the associated vector norm.

Proof

$$\begin{aligned} ||A^{-1}|| &= \max_{w\neq 0}\left\{\frac{||A^{-1}w||}{||w||}|w\in\mathbb{R}^n\right\} \\ &= \max_{w\neq 0}\left\{\frac{||u||}{||Au||}|u\in\mathbb{R}^n\right\} \leq \frac{||u||}{\alpha||u||} = \frac{1}{\alpha} \end{aligned}$$

□

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:

$$\begin{aligned} ||u|| &\leq ||(I+B)u|| + ||Bu|| \\ ||(I+B)u|| &\geq ||u|| - ||Bu|| \\ &\geq ||u|| - ||B||\,||u|| \\ &\geq (1-||B||)||u|| \\ &\geq \alpha||u|| \end{aligned}$$

□

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

$||u||_2^2 = u^Tu$

For example:

if $\exists\, \alpha > 0$ with $u^T Au \geq \alpha u^T u \quad \forall u$ then A is invertible with

$$||A^{-1}|| \leq \frac{1}{\alpha}$$
$$\alpha||u||_2^2 \leq u^T Au$$
$$\alpha||u||_2^2 \leq u^T Au \leq ||u||_2||Au||_2 \Rightarrow \alpha||u||_2 \leq ||Au||_2$$

□

Note: if $A = A^T$ and satisfies $u^T Au \geq \alpha||u||_2^2 \quad \alpha > 0, \forall u$, 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 square matrix

$$p(\lambda) \equiv \det(A - \lambda I) \quad \text{"characteristic polynomial of } A''$$

if λ is zero of $p \Rightarrow (A - \lambda I)\mathbf{x} = \mathbf{0}$ has nontrivial solution. These zeros are eigenvalues of A, $\mathbf{x}$ are eigenvectors

if λ eigenvalue $\Rightarrow A\mathbf{x} = \lambda\mathbf{x}$ so A takes $\mathbf{x}$ into a scalar multiple of itself. Suppose λ 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.

definition: The set of all eigenvalues λ of A is 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: $A \in M_n$

(i) $\left[\rho(A^TA)\right]^{\frac{1}{2}} = ||A||_2 \to ||A||_2 \equiv \max_{||x||_2=1} ||Ax||_2$

(ii) $\rho(A) \leq ||A|| \quad ||\cdot||$ induced norm.

example

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 3 & 2 & -1 \\ 2 & 6 & 4 \\ -1 & 4 & 5 \end{bmatrix}$$

$$\begin{aligned} \det(A^TA - \lambda I) &= \det \begin{bmatrix} 3-\lambda & 2 & -1 \\ 2 & 6-\lambda & 4 \\ -1 & 4 & 5-\lambda \end{bmatrix} \\ &= -\lambda^3 + 14\lambda^2 - 42\lambda \\ \lambda &= 0, \quad \lambda = 7 \pm \sqrt{7} \\ ||A||_2 &= \sqrt{\rho(A^TA)} = \sqrt{\max(0, 7-\sqrt{7}), 7+\sqrt{7})} \approx 3.106 \end{aligned}$$

□

We'll need this later:

definition: $A \in M_n$ convergent if

$$\lim_{k\to\infty}(A^k)_{ij} = 0 \quad \begin{array}{l} \text{for each } i = 1, 2, \ldots, n, \\ j = 1, 2, \ldots, n. \end{array}$$

Example

$$A = \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{2} \end{bmatrix} \Rightarrow A^2 = \begin{bmatrix} \frac{1}{4} & 0 \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}, A^3 = \begin{bmatrix} \frac{1}{8} & 0 \\ \frac{3}{18} & \frac{1}{8} \end{bmatrix} A^4 = \begin{bmatrix} \frac{1}{16} & 0 \\ \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

$$A^k = \begin{bmatrix} \left(\frac{1}{2}\right)^k & 0 \\ \frac{k}{2^{k+1}} & \left(\frac{1}{2}\right)^k \end{bmatrix} \quad \text{Since} \begin{cases} \lim\limits_{k\to\infty} \left(\frac{1}{2}\right)^k = 0 \\ \lim\limits_{k\to\infty} \frac{k}{2^{k+1}} = 0 \end{cases}$$

A is convergent. Note $\rho(A) = \frac{1}{2}$, since $\dfrac{1}{2}$ is only eigenvalue of A.

□

Theorem: The following statements are equivalent

(i) A is convergent

(ii) $\lim_{n\to\infty} ||A^n|| = 0$ for any induced norm.

(iii) $\lim_{n\to\infty} ||A^n|| = 0$ $\forall$ induced norms.

(iv) $\rho(A) < 1$

(v) $\lim_{n\to\infty} A^n\mathbf{x} = 0$ $\forall\mathbf{x}$

□

11.2.1 Matrix Norm, spectral radius and Condition Number

The notion of condition number is defined in terms of the matrix norm(s), so clearly the two notions are related. The matrix norm measures the 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 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.

Lets 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 b_i, $i = 1, \ldots, n$. Let λ_i be the corresponding eigenvalues so that $Bb_i = \lambda_i b_i$. Note that in this case where $B = A^T A$, the λ_i must be non-negative real numbers, since we can write $b_i^T B b_i = b_i^T \lambda_i b_i = \lambda_i$ and also $b_i^T B b_i = b_i^T A^T A b_i = (Ab_i)^T Ab_i = ||Ab_i||_2^2 \geq 0$. Using this basis, we can compute the square of the 2-norm of A. The set of unit vectors $x|||x||_2 = 1$ can also be written as $x|x = \sum_{i=1}^n y_i b_i$, 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 $x^T(A^T A)x = x^T Bx = \sum_{i=1}^n \lambda_i y_i^2$.

We can now rewrite the 2-norm squared of A as

$$||A|||_2^2 = max_{y_i|\sum y_i^2=1} \sum_{i=1}^n \lambda_i y_i^2.$$

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

$$\text{For symmetric matrices} ||A||_2 = \rho(A).$$

Example This last relation is definitely false for general matrices A. For example, consider

$$A = \begin{pmatrix} 1 & 1000 \\ 0 & 1 \end{pmatrix}$$

, The eigenvalues of A are both 1, so $\rho(A) = 1$, however,

$$A^T A = \begin{pmatrix} 1 & 1000 \\ 1000 & 1000001 \end{pmatrix}$$

has eigenvalues of 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.

The condition number (in the 2-norm) for the A of this example 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 $||A||_2^2 \approx 1000001.999999$

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 λ and using the definition of a matrix norm.

$$||A|| = max_{x|||x||=1}||Ax|| \geq ||Av|| = ||\lambda v|| = |lambda|||v|| = |lambda|$$

Thus the matrix norm is always larger than the maximum of the absolute values of the eigenvalues, i.e. the spectral radius. The careful reader will notice that there is the possibility that λ 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 etc. still work. They do.

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_∞ norm, $||x|| \leq 1$ is a square, S_1 centered at 0 with sides parallel to the axes of length 2, and the matrix ∞-norm is the half the side

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 ∞-norm are illustrated in the figure below for the matrix $A = \begin{pmatrix} 1 & 4 \\ 0 & 1 \end{pmatrix}$.

□

11.3 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

$$T \equiv U^\dagger A U$$

is upper triangular. Note: the † indicates conjugate transpose.

Since T is upper triangular and $U^\dagger = U^{-1}$, the characteristic equation of A is

$$f_A(\lambda) = f_T(\lambda) = (\lambda - t_{11})(\lambda - t_{22}) \cdots (t - t_m)$$

$\therefore$ the eigenvalues of A are diagonal elements of T □

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

$$U^+ A U = D = \text{ diag } [\lambda_1, \lambda_2, \ldots, \lambda_n]$$

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.

Jordan Canonical Form:

definition: "Jordan block" is an $n \times n$ matrix

$$J_n(\lambda) = \begin{bmatrix} \lambda & 1 & 0 & \cdots & & 0 \\ 0 & \lambda & 1 & 0 & & \\ : & 0 & \lambda & 1 & & \\ : & & & & & \\ : & & & & \ddots & 1 \\ 0 & \cdots & & 0 & & \lambda \end{bmatrix} \qquad n \geq 1$$

where $J_n(\lambda)$ has the single eigenvalue λ 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

$$P^{-1}AP = \begin{bmatrix} J_{n_1}(\lambda_1) & & & & 0 \\ & J_{n_2}(\lambda_2) & & & \\ & & J_{n_3}(\lambda_3) & & \\ & & & \ddots & \\ 0 & & & & J_{n_r}(\lambda_r) \end{bmatrix}$$

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 hte geometric multiplicities must be n the order of matrix A.

□

Remark:

$$P^{-1}AP = D + N$$
$$D = \text{ diag } [\lambda_1, \ldots, \lambda_r]$$

with each λ_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

$$N^n = 0$$

□

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 $m \rightarrow \infty$ 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

$$(I - A)^{-1} = I + A + A^2 + \cdots + A^m + \cdots \tag{51}$$

Conversely, if (51) is convergent $\Rightarrow r_\sigma(A) < 1$.

□

Theorem: A as above. If $||A|| < 1 \Rightarrow (I - A)^{-1}$ exists and has the expansion (51). Moreover

$$||(I - A)^{-1}|| \leq \frac{1}{1 - ||A||}$$

□

Theorem: Let A and B be $n \times n$ matrices. Assume A nonsingular and suppose that

$$||A - B|| < \frac{1}{||A^{-1}||}$$

then B is also nonsingular

$$||B^{-1}|| \leq \frac{||A^1||}{1 - ||A^{-1}||\,||A - B||}$$

and

$$||A^{-1} - B^{-1}|| \leq \frac{||A^{-1}||^2\,||A - B||}{1 - ||A^{-1}||\,||A - B||}$$

□

11.4 Condition Number and Error Estimates:

Want to know how good a computed solution is

Theorem: Suppose $A \in M_n$ nonsingular and $\mathbf{x_c}$ approx to $\mathbf{x}_t$, the exact solution to $A\mathbf{x} = \mathbf{b}$, $\mathbf{b} \neq \mathbf{0}$. Then any compatible nroms (matrix and vector) give

$$(52) \quad \frac{1}{||A||\,||A^{-1}||}\frac{||\mathbf{A}\mathbf{x}_c - -\mathbf{b}||}{||\mathbf{b}||} \leq \frac{||\mathbf{x}_c - \mathbf{x}_t||}{||x_t||} \leq \underbrace{||A||\,||A^{-1}||}_{k(A)}\underbrace{\frac{||A\mathbf{x}_c - \mathbf{b}||}{||\mathbf{b}||}}_{\varepsilon}$$

Proof: Use (50)

□

definition: Condition Number $k(A) = ||A||\,||A^{-1}||$

Then

(52) $\frac{\varepsilon}{k(A)} \leq \frac{||\mathbf{x}_c - \mathbf{x}_z||}{||\mathbf{x}_z||} \leq \varepsilon k(A)$

$k(A) \geq 1$ and the closer $k(A)$is to 1 the more accurate ε becomes as a measurement of the error.

if $k(A) \gg 1 \Rightarrow$ relative error may not be small even if ε is small.

When $k(A) \gg 1$ it indicates $A\mathbf{x} = \mathbf{b}$ is ill-conditioned, i.e. small changes input data produce large changes in output $\mathbf{x}$. Ultimate ill-conditioning: when A is singular (in this case there are infinitely many vectors $\mathbf{u} \neq \mathbf{0}$ such that $A\mathbf{u} = \mathbf{0}$, and hence $\mathbf{x}_t$ and $\mathbf{x}_t + \mathbf{u}$ are both solutions of $A\mathbf{x} = \mathbf{b}$ even when A and B undergo no change at all.

One can show that the connection between size of $k(A)$ and how close A is to being singular, if A is nonsingular

$$\frac{1}{k(A)} = \min\left\{\frac{||A - B||}{||A||} : B \text{ is singular}\right\}$$

$\therefore$ A can be approx by singular matrix if and only if $k(A)$is large.

Remark: So we want $\mathbf{x}_c$ and an estimate of $k(A)$ to see how go solution is. Don't want to find A^{-1}. Could estimate as follows:

$$\begin{aligned} \text{for } \mathbf{x} \neq \mathbf{0} \quad & \mathbf{x} = A^{-1}A\mathbf{x} \\ & ||\mathbf{x}|| \leq ||A^{-1}||\,||A\mathbf{x}|| \therefore \\ & ||A^{-1}|| \geq \frac{||\mathbf{x}||}{||A\mathbf{x}||} \end{aligned}$$

so estimate $||A^{-1}||$ by making $||\mathbf{x}||/||A\mathbf{x}||$ as large as possible which vector $\mathbf{x}$ to choose? depends on a problem

□

11.5 EIGENVALUES AND THE CANONICAL FORMS OF MATRICES (Graduate Students)

Mostly taken from Atkinson's Numerical Analysis Book.

The statement

$$Ax = \lambda x$$

with $x \neq 0$

λ complex or real, $x \in \mathbb{C}^n$, $A \in \mathbb{C}^{n\times n}$ is an eigenvalue problem.

$\lambda \equiv$ eigenvalues, $x \equiv$ eigenvectors.

λ 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 λ 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

$$A = Q^{-1}BQ \text{ where } Q = P^{-1}.$$

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:

A is $n \times n$ matrix $\quad A = [a_{ij}] \quad i, j = 1, \ldots, n.$

Let

$$r_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad i = 1, 2, \ldots, n. \tag{53}$$

Let Z_i denote the circle in the complex plane with center a_{ii} and radius r_i:

$$Z_i = \{z \in \mathbb{C} \text{ such that } |z - a_{ii}| \leq r_i\} \tag{54}$$

Theorem (Gerschgorin): A as above. Let λ be an eigenvalue of A then λ 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 $\Rightarrow$ these results are also valid if summation within the column rather than in the row is used defining the radii in (53)

Proof: See Figure 53.

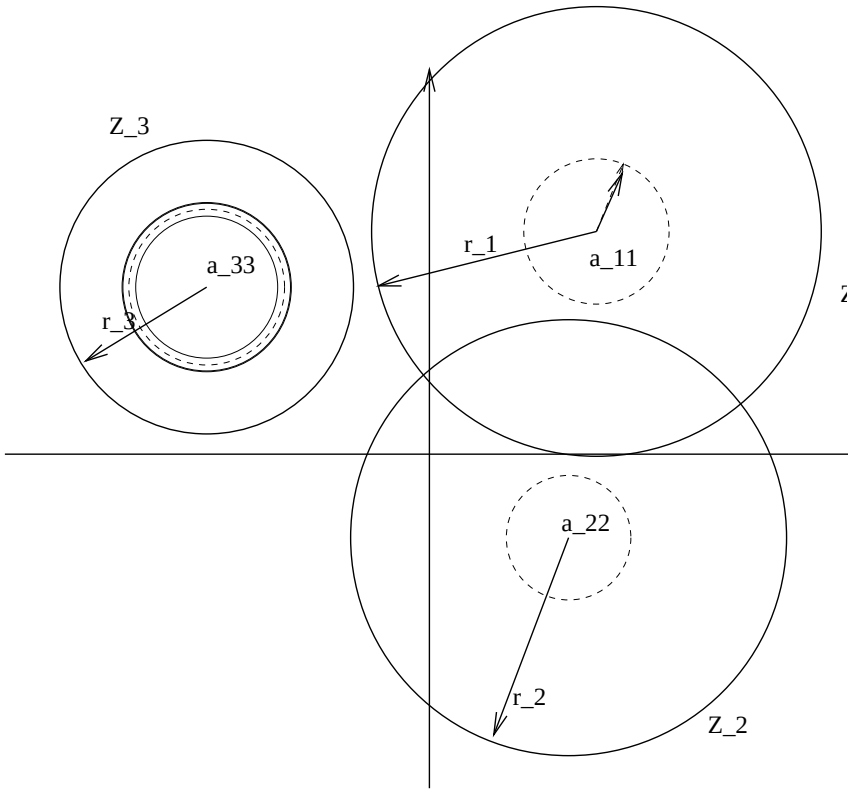

Figure 53: Example of Gerschgorin's circle theorem. Solid circles given by (54). According to theorem there should be 1 eigenvalue in Z_3 and two in the union of Z_1 and Z_2.

Let λ be an eigenvalue of A and x its corresponding eigenvector. Let k be such that

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

then from $Ax = \lambda x$, the k^{th} component

$$\sum_{j=1}^{n} a_{kj}x_j = \lambda x_k$$

$$(\lambda - a_{kk})x_k = \sum_{\substack{j=1\\k\neq j}}^{n} a_{kj}x_j$$

$$|\lambda - a_{kk}||x_k| \le \sum_{\substack{j=1\\k\neq j}}^{n} |a_{jk}||x_j| \le r_k||x||_\infty$$

Canceling $||x||_\infty$ proves first part of theorem.

Let $D = \text{diag}\,[a_{11}, a_{22}, \ldots, a_{nn}]$ and

$$E = A - D$$

for $0 \le \varepsilon \le 1$ let

(55) $$A(\varepsilon) = D + \varepsilon E$$

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 ε and since the roots of any polynomial are continuous functions of its coefficients $\Rightarrow \lambda(\varepsilon)$ are continuous functions of ε. As ε varies, each $\lambda_i(\varepsilon)$ changes in complex plane, marking a path from $\lambda_i(0)$ to $\lambda_i(1)$. From first part of the theorem

$$Z_i(\varepsilon) = \{z \in \mathbb{C} \text{ such that } |z - a_{ii}| \le \varepsilon r_i\} \quad i = 1, 2, \ldots, n.$$

with r_i defined as before. Examples of these circles are in figure. The circles decrease as ε 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 $\lambda_i(0)$.

□

Example) Consider

$$A = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 0 & -1 \\ 1 & 1 & -4 \end{bmatrix}$$

Theorem shows eigenvalues must be contained in the circles

$$|\lambda - 4| \leq 1 \quad |\lambda| \leq 2 \quad |\lambda + 4| \leq 2 \tag{56}$$

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 (56) 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 (55) 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

$$[-6, -2), \quad (-2, 2], \quad [3, 5]$$

the true eigenvalues are $-3.76010, -.442931, 420303$. □

Example) Consider the $n \times n$ matrix

$$A = \begin{bmatrix} 4 & 1 & 0 & & & & \cdots & & 0 \\ 1 & 4 & 1 & 0 & & & & & 0 \\ 0 & 1 & 4 & 1 & 0 & & & & \\ 0 & 0 & 1 & 4 & 1 & & & & \\ & & 0 & 1 & 4 & 1 & & & \\ & & 0 & 0 & 1 & 4 & 1 & & \\ & & & 0 & 0 & 1 & 4 & 1 & \\ & & & & & & \ddots & & \\ 0 & & & & & & 0 & 1 & 4 & 1 \\ 0 & \cdots & & & & & & 0 & 1 & 4 \end{bmatrix}$$

Since A is symmetric then all eigenvalues 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:

$$P^{-1}AP = \text{diag}[\lambda_1, \cdots, \lambda_n] = D \tag{57}$$

P nonsingular. The columns of P will be eigenvectors of A corresponding to eigenvalues $\lambda_1, \ldots, \lambda_n$. Matrices for which (57) holds occur often in practice ... but this is a special case.

Note: For diagonal matrices $G = \text{diag } [d_1, d_2, d_3, \ldots, d_n]$

$$||G|| = \max_{1 \leq 1 \leq n} |d_i|.$$

Theorem (Bauer-Fike) Let A be as above. Let $A + E$ be a perturbation of A and λ an eigenvalue of $A + E \Rightarrow$

$$\min_{1 \le i \le n} |\lambda - \lambda_i| \le ||P||\,||P^{-1}||\,||E|| \tag{58}$$

□

Corollary: If A is Hermitian and $A + C$ a perturbation of $A \Rightarrow$

$$\min_{i \le i \le n} |\lambda - \lambda_i| \le ||E||_2$$

for any eigenvalue λ of $A + E$. □

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 λ_i and $\hat{\lambda}_i$ $i = 1, 2, \ldots, n$ be eigenvalues of A and $\hat{A}$, arranged in increasing order. Then

$$\left[\sum_{j=1}^{n} (\lambda_j - \hat{\lambda}_j)^2\right]^{\frac{1}{2}} \le F(E) \tag{59}$$

where $F(E) = [\sum_{i,j=1}^{n} |a_{ij}|^2]^{1/2}$, the Frobenius norm.

□

A computable error bound for symmetric matrices

A symmetric $n \times n$. Let x and λ be approximated computed eigenvector and eigenvalue. Then let residual

$$\eta = Ax - \lambda x$$

Since A symmetric $\Rightarrow \exists$ a unitary matrix U

$$U^\dagger A U = \text{ diag}[\lambda_1, \lambda_2, \ldots, \lambda_n] \equiv D$$

then $\min_{1 \le i \le n} |\lambda - \lambda_i| \le \dfrac{||\eta||_2}{||x||_2}$

Exercise Show this.

□

11.5.3 Eigenvalues of 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

$$f_k(\lambda) = \begin{bmatrix} \alpha_1 - \lambda & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 - \lambda & \beta_2 & & : \\ 0 & & \ddots & & \\ : & : & & & \\ & & & & \beta_{k-1} \\ 0 & \cdots & 0 & \beta_{k-1} & \alpha_k - \lambda \end{bmatrix}$$

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 $2 \leq k \leq n$.

12 NUMERICAL LINEAR ALGEBRA

12.1 Direct Methods for Linear Systems

Let

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & & & \\ : & & & \\ a_{n1} & \cdots & & a_{nm} \end{bmatrix}.$$

A is $n \times m$ 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.

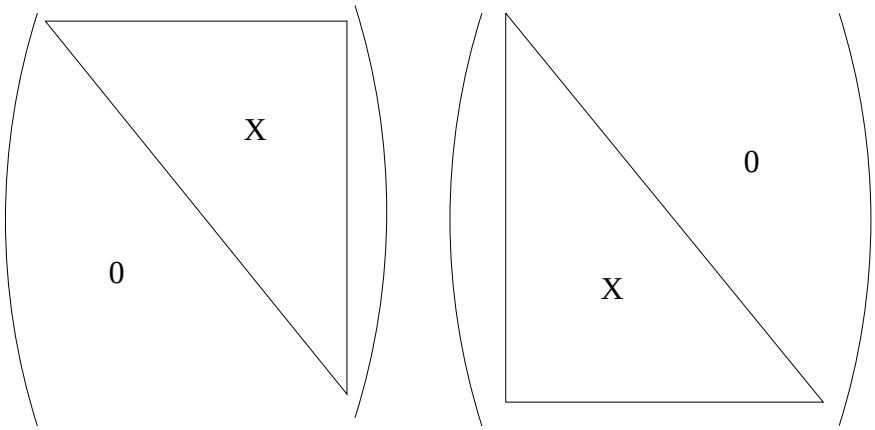

Figure 54: Upper and Lower triangular matrices. 0 indicates exclusively zero entries, X indicates not all zero entries.

Three operations are permitted to simplify a linear system. These are *Elementary Operations*:

(a) Row multiply by $\lambda \neq 0, \lambda$ is a scalar.

(b) $\lambda\times$ (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.

□

12.1.1 Gaussian with Back Substitution:

Use above rules to form a "reduced" or "triangular" equivalent matrix problem, of the form shown in Figure 54

Consider the linear system

$$x_1 - x_2 + 2x_3 - x_4 \;\; = \;\; -8$$

$$\begin{aligned} 2x_1 - 2x_2 + 3x_3 - 3x_4 &= -20 \\ x_1 + x_2 + x_3 &= -2 \\ x_1 - x_2 + 4x_3 + 3x_4 &= 12 \end{aligned}$$

expressed compactly as $Ax = b$. This system can be expressed in the following $n \times n + 1$ "Augmented Matrix"

$$\left(\begin{array}{cccc|c} \underline{1} & -1 & 2 & -1 & -8 \\ 2 & -2 & 3 & -3 & -20 \\ 1 & 1 & 1 & 0 & -2 \\ 1 & -1 & 4 & 3 & 12 \end{array}\right)$$

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

$$\left(\begin{array}{cccc|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & \underline{2} & -1 & 1 & 6 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right)$$

The underlined element is the new pivot. The next operation is to exchange E_2 and E_3, resulting in

$$\left(\begin{array}{cccc|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & \underline{-1} & -1 & -4 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right)$$

Next, take $2E_3 + E_4 = E_4$, yielding

$$\left(\begin{array}{cccc|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 0 & 2 & 4 \end{array}\right)$$

At this point we have an upper triangular matrix and back substitution, starting with E_4 going back up to E_1 gives us

$$\begin{array}{lll} 2x_4 = 4 & \Rightarrow x_4 = 2 & \\ -x_3 - x_4 = -4 & x_3 = 4 - x_4 = 2 & \\ 2x_2 - x_3 + x_4 = 6 & \Rightarrow 2x_2 - 2 + 2 = 6 & \Rightarrow x_2 = 3 \\ x_1 - x_2 + 2x_3 - x_4 = 8 & \Rightarrow x_1 - 3 + 4 - 2 = -8 & \Rightarrow x_1 = -7 \end{array}$$

□

Gaussian Elimination Algorithm(no pivoting)

Solves $A\mathbf{x} = \mathbf{b}$

Input: $A = (a_{ij})$ $\quad 1 \leq i \leq n \quad 1 \leq j \leq n+1$
A is the augmented matrix

Output: x_i, $1 \leq i \leq n$ or error message.

1. for $i = 1 \ldots n-1$ do steps 2-4. % elimination
2. Find p the smallest integer with $1 \leq p \leq n$ and $a_{pk} \neq 0$. If no p can be found, output ('No unique solution')
3. if $p \neq k$ then $\{E_p \leftrightarrow E_i\}$ % swap if needed
4. for $j = i+1 \ldots n$ do steps 5 and 6.
5. $m_{ji} = a_{ji}/a_{ii}$;
6. $E_j - m_{ji}E_i \leftrightarrow E_j$ % elimination
7. if $a_{nn} = 0$ output ('no unique solution') STOP
8. $x_n = a_{n,n+1}/a_{nn}$ % start of backward substitution
9. for $i = n-1 \ldots 1$
   $x_i = [a_{i,n+1} - \sum_{j=i+1}^{n} a_{ij}x_j]/a_{ii}$
10. output (x , the solution)

□

What types of failures could we expect? Here are two examples:

Example

$$A = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 6 \\ 1 & 1 & 2 & 6 \end{array}\right] \quad B = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 2 & 2 & 1 & 4 \\ 1 & 1 & 2 & 6 \end{array}\right]$$

$a_{11} = 1 \Rightarrow (E_2 - 2E_1) \to E_2$ and $E_3 - E_1 \to E_3$

$$\widetilde{A} = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 1 & 2 \end{array}\right] \quad \widetilde{B} = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -4 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

Note: $a_{22} = a_{32} = 0$

For $\widetilde{A}$ $x_3 = 2$, $-x_3 = -2$, $x_2 = 2 - x_1$ $\therefore \infty\#$ of solutions
For $\widetilde{B}$ $x_3 = 2$, $x_3 = 4$ $\therefore$ no solution.

□

Performance Issues:

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: only require 2D array of size $(n, n+1)$

since we can store m_{ji} in the 0 entries a_{ji} (below diagonal entries are not involved in the computation after pivoting).

Operation count:

Multiply/divide $\to$ 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 $\rightarrow$ short operations, since they take a shorter amount of compute time.

Look at Step 5 and 6 in the Gaussian Elimination algorithm without pivoting (see 12.1.1):

Step 5 $n-i$ divisions performed.

Step 6 The replacement $E_j - m_{ji}E_i) \rightarrow E_j$ requires m_{ji} multiplied by each term E_i, resulting in a total: $(n-i)(n-i+1)$ 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

$\therefore$ For each $i = 1, 2, \ldots, n-1$ the operations required in Steps 5 and 6 are:
Mult/Div $\quad (n-i) + (n-i)(n-i+1) = (n-i)(n-i+2)$

Adds/Subt $\quad (n-i)(n-i+1)$

Recall:

$$\sum_{j=1}^{n} 1 = n,$$

$$\sum_{j=1}^{m} j = \frac{m(m+1)}{2},$$

$$\sum_{j=1}^{n} j^2 = \frac{m(m+1)(2m+1)}{6}$$

therefore, summing over i:

Mult/Div: $\displaystyle\sum_{i=1}^{n-1}(n-i)(n-i+2) = (n^2+2n)\sum_{i=1}^{n-1} 1 - 2(n+1)\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} i^2 = \frac{2n^3+3n^2-5n}{6}$

Add/Sub: $\displaystyle\sum_{i=1}^{n-1}(n-i)(n-i+1) = \frac{n^3-n}{3}$

Now we go to the back substitution portion, Steps 8 and 9: Step 8: 1 division
Step 9: $n-i$ multiplies and $n-i-1$ adds for each summation term, then 1 subtract and 1 divide.

So the total operation count in Steps 8 and 9:

Mult/Div: $1 + \sum_{i=1}^{n-1}[(n-i)+1] = \frac{n^2+n}{2}$;

Add/Sub: $\sum_{i=1}^{n-1}[(n-i-1)+1] = \frac{n^2-n}{2}$

Total Operation count:

Mult/Div: $\frac{2n^3+3n^2-5n}{6} + \frac{n^2+n}{2} = n^2 - n/3 + \frac{n^3}{3}$

Add/Sub: $\frac{n^3-n}{3} + \frac{n^2-n}{2} = \frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$

Therefore this algorithm is an $O(n^3)$ operation. 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 better off with an iterative method (see 12.3).

12.1.2 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.

Example

$$\begin{aligned} 0.0001x + 1.00y &= 1.00 \\ 1.00x + 1.00y &= 2.00 \end{aligned}$$

True solution rounded to 5 significant digits is $x = 1.00010 \quad y = 0.99990$.

However, without rearrangement $-10000y = -10000 \Rightarrow y = 1.00$ and $x = 0.00$.

With rearrangement

$$\begin{aligned} 1.00x + 1.00y &= 2.00 \\ 0.0001x + 1.00y &= 1.00 \end{aligned}$$

$$\text{we get } x \;=\; 1.00 \quad y = 1.00$$

Great improvement!

Analysis of problem:

Example

$$\begin{aligned} E_1 : 0.003000x_1 + 59.14x_2 &= 59.17 \\ E_2 : 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

has exact solution $x_1 = 10.00 \quad x_2 = 1.00$

Use 4 digit rounding:

$$a_{11} = 0.003000 \text{ is pivot and small}$$
$$m_{21} = \frac{5.291}{a_{11}} = 1763.6\overline{6} \text{ rounds to } 1763.$$

$$\begin{aligned} &(E_2 - m_{21}E_1) \to E_2 \text{ gives} \\ &0.003000x_1 + 59.14x_2 = 59.17 \\ &-104300x_2 = -104400 \\ &\text{instead of precise value} \\ &0.003000x_1 + 59.14x_2 = 59.17 \\ &-104309.37\overline{6}x_2 = -104309.37\overline{6} \end{aligned}$$

$\to$ disparity in magnitude of $m_{21}b_1$ and b_2 has introduced roundoff, but error has not yet propagated. Backward substitution yields:

$$x_2 = 1.001 \text{ close to } x_2 = 1.000$$

However

$$x_1 \approx \frac{59.17 - (59.14)(1.001)}{0.003000} = -10.00$$

contains small error of 0.001 multiplied by $\dfrac{59.121}{0.003000}$. Geometrically, it is clear what is happening: see Figure 55

The problem can be seen most clearly as follows:

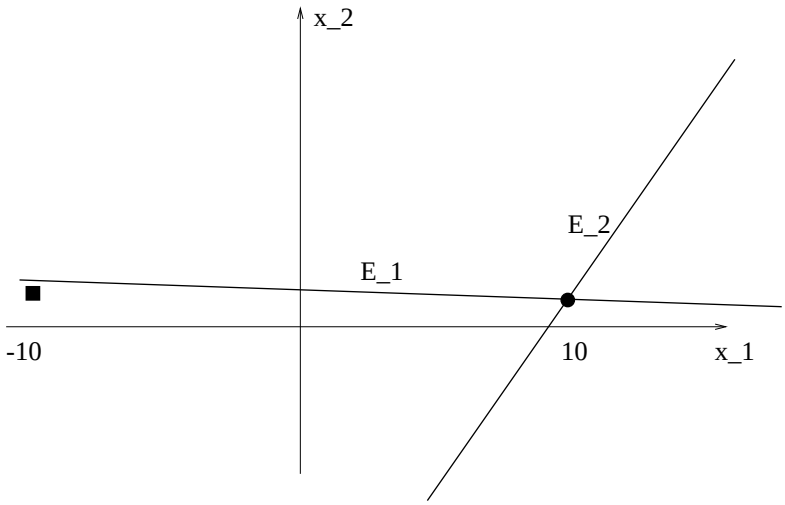

Figure 55: The solution is the point of intersection of the vectors E_1 and E_2. The predicted approximation, using 4 digit rounding is indicated by the square box.

if $|a_{11}|$ small compared to $|a_{i1}| \Rightarrow \lambda = -a_{i1}/a_{11}$ is very large.

then $(E_i + \lambda E_1) \to E_i$ gives
$(a_{i2} + \lambda a_{12})x_2 + \cdots (a_{in} + \lambda a_{in})x_n = b_i + \lambda b_1$

if λ large, it's like replacing the E_i by a multiple of E_i!

If λ small, then no problem, although rounding still occurs.

Strategy: Find row index I such that

$$|a_{I1}| = \max_{1 \le i \le n} |a_{i1}|$$

and rewrite the system so that I^{th} equation becomes first equation.

Now $-a_{i1}/a_{I1}$ is small.

Process is repeated...this is called *Partial Pivoting* and it involves row exchanges only.

Partial Pivoting as opposed to Full pivoting...it comes from direct application of the "elementary transformations", but is done in a smart way. It is a simple method, as compared to full pivoting, discussed later.

In full pivoting further reduction can be accomplished, however, it is more complex since it involves both row and column exchanges.

Look at the following partial pivoting example:

$$\left[\begin{array}{cc|c} 0.003000 & 59.14 & 59.17 \\ 5.291 & -6.130 & 46.78 \end{array}\right]$$

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 $m_{21} = \dfrac{a_{21}}{a_{11}} = 0.0005670$ small.

For larger systems look for largest $|a_{i1}\,|1 \leq i \leq n$. Take

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ : & : & : & : & : \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right]$$

suppose it's a_{21} then "exchange row E_{I+1} with E_2." Use a_{21} for pivot value and perform row reduce.

$$\left[\begin{array}{cccc|c} a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ 0 & \times & \cdots & \times & \times \\ 0 & \times & \cdots & \times & \times \\ 0 & \times & \cdots & \times & \times \\ 0 & \times & \cdots & \times & \times \end{array}\right]$$

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 12.1.6.

Another strategy: same thing but also perform column exchanges. **Total** or **Full** pivoting: expensive because it is hard to keep track of row and column exchanges. This ensures that pivot is maximal!

First step find $|a_{IJ}| = \max_{1\le i,j\le n} |a_{ij}|$. Retain the I^{th} equation and eliminate the coefficient of x_J in the I^{th} equation, $1 \le i \le n, i \ne I$ by replacing the i^{th} equation by $(-a_{iJ}/a_{IJ}) \times E_I + E_i \to E_i$

□

Pivoting may still fail:

Example

$$\begin{aligned} E_1 : 30.00x_1 + 591400x_2 &= 591700 \\ E_2 : 5.291x_1 - 6.130x_2 &= 46.78 \\ m_{21} = \frac{5.291}{30.00} &= 0.1764 \\ \Rightarrow 30.00x_1 + 591400x_2 &\approx 591700 \\ -104300x_2 &\approx -104400 \\ \Rightarrow x_1 \approx -10.00 \quad x_2 &\approx 1.001 \Leftarrow \text{Wrong.} \end{aligned}$$

Problem always requires suspicion if entries are of disparate sizes. This can sometimes be ameliorated using a *scaling* strategy.

There are a number of scaling strategies. Consider here "Row Scaling"

Take $A\mathbf{x} = \mathbf{b}$
multiply $E_i$by $r_i \quad i = 1, \ldots n.$

$$A\mathbf{x} = \mathbf{b} \to D_1 A\mathbf{x} = D_1\mathbf{b} \quad D_1 = \begin{bmatrix} r_1 & & & & \\ & r_2 & & & \\ & & r_3 & & \\ & & & \vdots & \\ & & & & r_n \end{bmatrix}$$

$$\text{Let } B = D_1 A$$

Choose r_i so that $B = b_{ij}$ satisfies

$$\max_j |b_{ij}| \approx 1 \quad i = 1, 2, \ldots, n \tag{60}$$

(60) is satisfied if $r_i = 1/s_i$

$$s_i = \max_j |a_{ij}| \quad i = 1, 2 \ldots, n \tag{61}$$

(62) $$\therefore b_{ij} = a_{ij}/s_j \quad i,j = 1,2\ldots,n.$$

(62) can produce additional rounding errors: Use "scaled partial pivoting": If the choice of pivot elements with scaling is forced to remain the same as with no scaling $\Rightarrow$ solution of scaled = unscaled.

Scaled Partial Pivoting:

1) Before elimination, compute s_i using (61) 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 $0(n^2)$ operation, which we clearly want to avoid.

Example

Solve:
$$\left[\begin{array}{cc|c} 0.7000 & 1725 & 1739 \\ 0.4352 & -5.433 & 3.271 \end{array}\right]$$
using 4 digit rounding row sizes $s_1 = \max(0.7, 1725) = 1725 \quad s_2 = 5.433$

Since $|a_{11}|/s_1 = 0.0004 < |a_{21}|/s_2 = 0.0801$

$\Rightarrow$ exchange rows 1 and row 2.
$$\left[\begin{array}{cc|c} 0.432 & -5.433 & 3.271 \\ 0.700 & 1725 & 1739 \end{array}\right]$$
$m_2 = 0.700/0.4352 = 1.608 \Rightarrow$ $\left[\begin{array}{cc|c} 0.4352 & -5.433 & 3.271 \\ & 1734 & 1734 \end{array}\right]$

$x_2 = 1.000 \quad x_1 = 20.00$ which is good.

Exercise

Do above without scaled partial pivoting. □

12.1.3 The LU Factorization

We saw $A\mathbf{x} = \mathbf{b}$ requires $\mathrm{O}(n^3)$ steps to solve by Gaussian elimination. Some matrices can be factored as

$$A = LU$$

$$\begin{aligned} \text{so } A\mathbf{x} = \mathbf{b} &\rightarrow LU\mathbf{x} = \mathbf{b} \\ \text{let } \mathbf{y} &= \mathbf{U}\mathbf{x} \\ \text{then } L\mathbf{y} = \mathbf{b} \quad &\text{since } L, U \text{ are trianglular} \\ &\text{they require } O(n^2) \text{ operations} \\ \therefore \text{ reduce } O(n^3) &\text{ to } O(n^2) \text{ to solve } L\mathbf{y} = \mathbf{b} \end{aligned}$$

□

Example $n = 100 \Rightarrow n^3 = 10^6 \quad n^2 = 10^4 \therefore \dfrac{n^2}{n^3} = 10^{-2}$,i.e. the reduction is 99%!

Of course to determine LU is still $\mathrm{O}(n^3)$.

□

To illustrate: assume Gaussian can be done without pivoting: i.e. nonzero pivot elements for each $i = 1, 2 \cdots, n$.

1) First step in Gaussian consists

$$(E_j - m_{j1}E_1) \rightarrow E_j \text{ where } m_{j1} = \frac{a_{j,1}^{(1)}}{a_{1,1}^{(1)}}$$

Get new matrix

$$A^{(2)} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots \\ : & : & a_{33}^{(2)} & \cdots \\ : & : & : & \cdots \\ : & 0 & : & \cdots \end{bmatrix}$$

$$A^{(2)}\mathbf{x} = M^{(1)}A\mathbf{x} = M^{(1)}\mathbf{b} = \mathbf{b}^{(2)}$$

$$M^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & 0 & 0 & \cdots & 0 \\ -m_{31} & 0 & 1 & 0 & 0 & \cdots & 0 \\ : & 0 & 0 & 1 & 0 & \cdots & 0 \\ : & : & : & : & \ddots & 0 & 0 \\ : & 0 & 0 & 0 & \cdots & 1 & 0 \\ -m_{n1} & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Construct $M^{(2)}$ in same way with $m_{j,2} = \frac{a_{j,2}^{(2)}}{a_{2,2}^{(1)}}$

$$A^{(3)}\mathbf{x} = M^{(2)}A^{(2)}\mathbf{x} = M^{(2)}M^{(1)}A\mathbf{x} = M^{(2)}M^{(1)}\mathbf{b} = \mathbf{b}^{(3)}$$

$\therefore$ if have $A^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$then multiply (left) by

$$M^{(k)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -m_{k+1,k} & 1 & 0 & \cdots & 0 \\ : & : & : & 0 & \ddots & 0 & 0 \\ 0 & 0 & -m_{n-1,k} & 0 & \cdots & 1 & 0 \\ 0 & 0 & -m_{n,k} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

the "Gaussian transformation matrix",

to get $A^{(k+1)}\mathbf{x} = M^{(k)}A^{(k)}\mathbf{x} = M^{(k)}\mathbf{b}^{(k)} = \mathbf{b}^{(k+1)} = M^{(k)}\cdots M^{(1)}\mathbf{b}$

Process ends with $A^{(n)}\mathbf{x} = \mathbf{b}^{(n)}$

$$A^{(n)} = M^{(n-1)}M^{(n-2)}\cdots M^{(1)}A \quad \text{upper triangle.}$$

$$A^{(n)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & a_{22}^{(2)} & & : \\ : & : & : & : \\ : & 0 & a_{n-1,n}^{(n-1)} & : \\ 0 & 0 & 0 & a_{nn}^{(n)} \end{bmatrix} = U$$

Now we tackle the L part:

1) Recall that $M^{(k)}$ is used to

$$A^{(k+1)}\mathbf{x} = M^{(k)}A^{(k)}\mathbf{x} = \mathbf{b}^{(k+1)}$$

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 $M^{(k)}$

$$L^{(k)} = [M^{(k)}]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & m_{k+1,k} & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & 0 & \ddots & 0 & 0 \\ 0 & 0 & m_{n-1,k} & 0 & \cdots & 1 & 0 \\ 0 & 0 & m_{n,k} & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Let $L = L^{(1)}L^{(2)}\cdots L^{(n-1)} = \begin{bmatrix} 1 & 0 & & \cdots & 0 \\ m_{21} & 1 & & & 0 \\ \vdots & & 1 & \cdots & \vdots \\ \vdots & & & 1 & 0 \\ m_{n,1} & \cdots & & m_{n,n-1} & 1 \end{bmatrix}$

The product of L with $U = M^{(n-1)}\cdots M^{(2)}M^{(1)}A$ gives

$$\begin{aligned} LU &= L^{(1)}L^{(2)}\cdots L^{(n-3)}L^{(n-2)}\underbrace{L^{(n-1)}\cdot M^{(n-1)}}_{I}M^{(n-2)}M^{(n-3)}\cdots M^{(2)}M^{(1)}A \\ &= L^{(1)}L^{(2)}\cdots L^{(n-3)}L^{(n-2)}IM^{(n-2)}M^{(n-3)}\cdots A \\ &= L^{(1)}L^{(2)}\cdots L^{(n-3)}IM^{(n-3)}\cdots A = A \end{aligned}$$

Theorem: If Gaussian elimination can be performed in $A\mathbf{x} = \mathbf{b}$ 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 12.1.1 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 $(L')-1L = D^{-1}D''$ 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.

□

Not every matrix can be LU-decomposed:

Example:

$$\begin{aligned} 4x_2 &= 8 \\ 4x_1 + 2x_2 &= 17 \end{aligned}$$

$$A = \begin{bmatrix} 0 & 4 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \ell_{21} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

$$u_{11} = 0$$

$$\ell_{21}u_{11} = 4 \therefore \text{ cannot satisfy.}$$

$$A = \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix}$$

this ensures that no u_{rr} is O. This rearrangement is always possible if the solution is unique.

□

with this understood, we can very quickly discuss three popular LU decomposition algorithms:

12.1.4 Doolittle, Crout's, and Choleski Algorithms

Doolittle Algorithm:	get $\ell_{ii} = 1$	$i = 1 \ldots n.$	"Unit lower Trianglular"
Crout's Algorithm:	get $u_{ii} = 1$	$i = 1 \ldots n$	"Unit Upper Triangular"
Choleski:	$U = L^T$	$\ell_{ii} = u_{ii}$	i=1...n

12.1.5 LDL factorization of symmetric matrices and Cholesky factorization of Positive definite matrices

One of the important themes 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 2×2 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 times the transpose of the

lower triangular matrix. In this case the diagonal entries are positive, so we can factor further as

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ 1/4 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 1 & 1/4 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 0 \\ 1/2 & \sqrt{7}/2 \end{pmatrix} \begin{pmatrix} 2 & 1/2 \\ 0 & \sqrt{7}/2 \end{pmatrix} \end{aligned}$$

That is we factor $A = LL^T$ with L a lower triangular matrix.

The same sorts of factorizations are possible for $n \times n$ matrices.

Before we dig into 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 (nonsingular) 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's 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

```
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 $x \neq 0$, $x^T Ax > 0$.

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 $x^T Ax = x^T LL^T x$. Let $u = L^T x$, then we have $x^T Ax = u^T u = \sum_{i=1}^{n} u_i^2$, and we have that unless $u = 0$, $x^T Ax > 0$. The only way $u = L^T x$ could be 0 is if $x = 0$, since L is nonsingular, so if $x \neq 0$, $x^T Ax > 0$.

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, $e_1^T Ae_1 = \alpha > 0$ 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 $x^T = (-u^T v/\alpha, u^T)$.

$$x^T A x = (-u^T v/\alpha, u^T) \begin{pmatrix} 0 \\ -(u^T v)v/\alpha + Cu \end{pmatrix} = u^T A' u,$$

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 Golub and Van Loan.

Remark: Note that this never accesses any elements above the diagonal. This is possible because the matrix is symmetric. We always access the lower triangle. This frees 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

$$A_1 = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ 1 & 4 & \cdot & \cdot \\ 4 & -4 & 5 & \cdot \\ -1 & 0 & 0 & 9 \end{bmatrix}.$$

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

$$A_2 = \begin{bmatrix} 1 & \cdot & \cdot & \cdot \\ 1 & 2 & \cdot & \cdot \\ 4 & -2 & 1 & \cdot \\ -1 & 0 & 0 & 9 \end{bmatrix}.$$

For $k = 3$ we have

- $A_3(3,3) = \sqrt{A_2(3,3)}$
- $A_3(4,3) = A_2(4,3)/A_3(3,3)$
- $A_3(4,4) = A_2(4,4) - A_3(4,3)A_3(4,3)$

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

$$LL^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 4 & -2 & 1 & 0 \\ -1 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 4 & -1 \\ 0 & 2 & -2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 4 & -1 \\ 1 & 5 & 0 & -1 \\ 4 & 0 & 21 & -4 \\ -1 & -1 & -4 & 10 \end{bmatrix} = A$$

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.

12.1.6 Permutation Matrix

Remark: Row exchange is needed to control round-off error, so it may seem impractical to use these algorithms. However, there are a number of matrix types where solution without row interchange is possible. First, let's present a useful construct, the Permutation matrix.

$(n \times n)$ 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 $P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

left-multiply:

$$PA = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Get different answer for right multiply AP.

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 $P^{-1} = P^T$

Remark: Since for any nonsingular matrix $A \Rightarrow A\mathbf{x} = \mathbf{b}$ can be solved by Gaussian elimination, with the possibility of row interchanges, then there is

a rearrangement that could result in a matrix that can be solved without row interchanges.

Thus with a non-singular there exists

$$PA\mathbf{x} = P\mathbf{b}$$

which can be solved without row interchanges.

$$\begin{aligned} PA = LU. \text{ Since } P^{-1} = P^T \\ \Rightarrow A = (P^T L)U \end{aligned}$$

However, unless $P = I$ $\quad P^T L$ is not lower triangular.

□

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 since $a_{11} = 0$. However, let's use the permutation matrices to transform the problem:

$E_1 \leftrightarrow E_2$, followed by $E_3 - E_1 \rightarrow E_3, \quad E_4 - E_1 \rightarrow E_4 \therefore$

$$\begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & -1 & 0 & 4 \end{bmatrix}$$

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 $(E_4 - E_3) \to E_3$

$$U = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Permutations associated with $E_1 \leftrightarrow E_2$ and $E_2 \leftrightarrow E_4$ are

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$P_1 : E_1 \leftrightarrow E_2$

$$P_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$P_2 : E_2 \leftrightarrow E_3$

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\therefore P = P_2 P_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } P^{-1} = P^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$PA = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 1 & 3 & 2 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}$$

OK, now we do Gaussian elimination if we were solving for $A\mathbf{x} = \mathbf{b}$ on PA:

$$\begin{matrix} E_2 - E_1 \to E_2 \\ (\alpha) \end{matrix} \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}$$

$$E_4 - E_1 \to E_4 \quad (\beta) \qquad \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 4 \end{bmatrix}$$

$$E_4 + E_2 \to E_4 \quad (\gamma) \qquad \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 5 \end{bmatrix}$$

$$E_4 - E_3 \to E_4 \quad (\delta) \qquad \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} = U \tag{63}$$

So to write down the LU factorization of PA:

from (α) we see that

$$L = \begin{bmatrix} 1 & 1 & 0 & 0 \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

from (β), (γ) and (δ) we see that

$$L = \begin{bmatrix} 1 & 1 & 0 & 0 \\ & & & \\ 1 & -1 & 1 & 1 \end{bmatrix}.$$

Since rows 1 and 3 are left unchanged

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{bmatrix}$$

$$\therefore PA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} \equiv LU$$

Multiply by $P^T \Rightarrow A = (P^T L)U$ and we recover A.

$\therefore$ Gaussian elimination can be performed on PA without row interchange using $E_2 - E_1 \to E_2$, $E_4 - E_1 \to E_4$, $E_4 + E_2 \to E_4$, $E_4 - E_3 \to E_4$

$$PA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 4 \end{bmatrix} = LU$$

□

What types of Matrices permit Gaussian Elimination without Row interchange? One important class is *strictly diagonally dominant* matrices.

definition: Strictly Diagonally Dominant $n \times n$ matrix A if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{u} |a_{ij}| \text{ holds for } i = 1, 2 \ldots n$$

Example

$$A = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{bmatrix}$$

$$B = \begin{bmatrix} 6 & 4 & -3 \\ 4 & -2 & 0 \\ -3 & 0 & 1 \end{bmatrix}$$

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 $B^T = B$

□

Theorem The following statements are equivalent for $n \times n$ matrix A:

a) $A\mathbf{x} = 0$ has a unique solution $\mathbf{x} = 0$.

b) Linear system $A\mathbf{x} = \mathbf{b}$ has unique solution for any n-column vector b.

c) A nonsingular $\Rightarrow A^{-1}$ exists

d) $\det A \neq 0$

e) Gaussian elimination with row exchanges can be performed on $A\mathbf{x} = \mathbf{b}$

□

12.2 Special Matrix Types

Theorem: A strictly diagonally dominant matrix A is nonsingular. Moreover, Gaussian elimination can be performed without row interchanges on $A\mathbf{x} = \mathbf{b}$ 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 $\mathbf{b}$ and $\mathbf{x}$ are n-dimensional vectors.

Take $A\mathbf{x} = \mathbf{0}$ and suppose that non-zero $\mathbf{x}$ 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

$$\text{when } i = k \quad a_{kk}x_k = -\sum_{\substack{j=1 \\ j\neq k}}^{n} a_{kj}x_j$$

this implies that

$$|a_{kk}||x_k| \leq -\sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{kj}||x_j|$$

$$\text{or } |a_{kk}| \leq \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{kj}|\frac{|x_j|}{|x_k|} \leq \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{kj}|$$

this contradicts the strict diagonal dominance of A therefore the only solution of $A\mathbf{x} = \mathbf{0}$ is $\mathbf{x} = \mathbf{0}$.

Proof: Next, prove that Gaussian elimination can be performed without row exchanges: show that $A^{(2)} \dots A^{(n-1)}$ generated by Gaussian process are strictly diagonal dominant.

Recall that a symmetric matrix A is positive definite if $\mathbf{x}^T A\mathbf{x} > 0$ for every $\mathbf{x}$, an n-dimensional vector.

$$\mathbf{x}^T A\mathbf{x} = [x_1 \dots x_n]^T [a_{ij}] \begin{bmatrix} x_1 \\ : \\ x_n \end{bmatrix} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

a single 1×1 matrix.

Not easy to check this way. Instead:

Theorem if A is positive definite $n \times n$ matrix then

a) A is nonsingular

b) $a_{ii} > 0$ for each $i = 1, 2, n$

c) $\max\limits_{1 \le k,j \le n} |a_{kj}| \le \max\limits_{1 \le i \le n} |a_{ii}|$

d) $(a_{ij})^2 < a_{ii} a_{jj}$ for each $i \neq j$.

Theorem: A is symmetric positive definite if and only if Gaussian elimination can be performed without row exchange on $A\mathbf{x} = \mathbf{b}$ 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 $l_{ii} \neq 0$.

□

definition: $n \times n$ 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 is then defined as the number $bw \equiv p + q - 1$. For example the matrix with non-zero entries marked as x:

$$A = \begin{bmatrix} x & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

has a bandwidth of $bw = 6$.

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. Furthermore, many matrices with small bandwidths have, in addition, other nice properties, for example, strong diagonal dominance, etc. Researchers spend much time optimizing codes that are storage efficient and also have nice mathematical properties which enable them to produce more efficient solvers. The efficient storage of matrices is called "Packing".

Example: Two common small bandwidth matrices which come up in the solution of pde's, for example, is the tri-diagonal and penta-diagonal matrices.

Tridiagonal Matrix (Banded Matrix) will have the structure:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & : \\ : & 0 & a_{43} & a_{44} & a_{44} & : \\ : & : & : & : & : & : \\ : & : & : & : & : & 0 \\ 0 & : & : & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

here $p = q = 2$, thus $bw = 3$.

□

If this matrix is diagonally dominant, then it is very easy to solve the problem $A\mathbf{x} = \mathbf{b}$, for $\mathbf{x}$: 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 $L\mathbf{y} = \mathbf{b}$, the the backward problem is to solve $U\mathbf{x} = \mathbf{y}$. These are two simple solves, since the matrices are triangular. Generalizations of the following algorithm exist for the penta-diagonal case, for the tri- or penta-diagonal case with 1 entry in the upper right hand corner and one in the lower left hand corner (see tridiagonal and pentadiagonal solvers at netlib. These occur frequently in the solution of pde's with periodic boundary conditions) .

12.2.1 Crout Factorization of Tri-diagonal Linear System

$A\mathbf{x} = \mathbf{b}$, with equations of the form

$$\begin{array}{lll} E_1 & a_{11}x_1 + a_{12}x_2 & = b_1 = a_{1,n+1} \\ E_2 & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 & = b_2 = a_{2,n+1} \\ : & : & = \quad : \\ E_n & a_{n,n-1}x_{n-1} + a_{nn}x_n & = b_n = a_{n,n+1} \end{array}$$

Note: the array b is stored in the $n+1$ column of the extended matrix A, of size $n \times (n+1)$.

Input: n, entries of A. The vector $\mathbf{b}$ 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 $\ell_{i,i-1} = a_{i,i-1}$ %i^{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

□

12.3 Iterative Methods for Solving Algebraic Systems

Want to solve

$$A(v) - b = 0, \tag{64}$$

where A is a nonlinear discrete operator, b is a known vector, and the discrete solution is v.

Often the solution to (64) 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

$$P(w) - b = 0$$

Table 1: Common Examples of Defect Correction Iteration

$P(v_{n+1})$	METHOD
$A(v_n) + J_A(v_n)(v_{n+1} - v_n)$	Newton
Diagonal J_A	Jacobi
Lower triangular part of J_A	Gauss-Seidel
Lower triangular part of J_A plus first upper off-diagonal	Line Gauss- Seidel
Coarse grid operator plus relax using one of the above	Multigrid
Symmetric part of J_A	Concus-Golub-O'Leary
If $\Lambda = (I + \Delta t L_x + \Delta t L_y)$, 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.	ADI
LU, where L lower U upper triangular	incomplete LU

that approximates (64) and is easier to solve, the *defect correction algorithm* may be appropriate: Given a guess v_n near a root v_{n+1} of (64) we can expand this equation using a Taylor series to get:

$$\begin{aligned} 0 &= A(v_{n+1}) - b \\ &= A(v_{n+1}) - b + P(v_{n+1}) - P(v_{n+1}) \\ &= A(v_n) - b + P(v_{n+1}) - P(v_n) - (J_P - J_A)(v_{n+1} - v_n) + O(\epsilon^2), \end{aligned} \tag{65}$$

where $\epsilon = v_{n+1} - v_n$. The defect correction iteration is any $O(\epsilon)$ approximation to (65).

The simplest such iteration is

$$P(v_{n+1}) = P(v_n) = A(v_n) + b. \tag{66}$$

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 1 lists some of the more common applications of defect corrections.

The iteration of (66) can often be sped up by using a one-step acceleration

parameter ω to give

$$P(v_{n+1}) = P(v_n) - \omega_n[A(v_n) - b].$$

These accerated methods include SOR, dynamic alternating direction implicit methods, and damped Newton. Often a 2-step acceleration method

$$P(v_{n+1}) = b + \omega_n[P(v_n) - \alpha_n A(v_n)] + (1 - \omega_n)P(v_{n-1})$$

can speed up the convergence even more. These method include Chebyshev and Conjugate Gradient methods.

12.3.1 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[6]. Some of the most common methods are:

1. Steepest Descent
2. Conjugate Gradient (see 12.3.4)
3. Broyden and variable metric methods (see BFGS at netlib
4. Fixed Point Methods
5. Method of Weighted Residuals

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 $\mathbf{f}(\mathbf{x}) \in C^2$ be an n-dimensional vector of functions that depend on $\mathbf{x} \in \mathbb{R}^n$. The problem we want to solve is the root-finding problem

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

[6]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

That is,

$$\begin{aligned} &f_i(x_1, \ldots, x_n) = 0 \quad 1 \leq i \leq n \\ &\mathbf{x} = (x_1, x_2 \ldots, x_n)^T \\ &\mathbf{f} = (f_1, f_2, \ldots, f_n)^T. \end{aligned}$$

Just as we did in the scalar case, we expand $\mathbf{f}$ about $\mathbf{x}$, locally (see 5.3) and retain to first order in $||\mathbf{H}||$:

$$\mathbf{0} = \mathbf{f}(\mathbf{x} + \mathbf{H}) \approx \mathbf{f}(\mathbf{x}) + J(\mathbf{x})\mathbf{H} + \mathcal{O}(||\mathbf{H}||^2).$$

Here, the *Jacobian Matrix* is

$$J(\mathbf{x}) \equiv \mathbf{f}'(\mathbf{x}) = \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \cdots & & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \cdots & \cdots & \partial f_2/\partial x_n \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \cdots & \partial f_n/\partial x_n \end{bmatrix}.$$

Hence, we can solve formally for the perturbation:

$$\mathbf{H} = -\left[J(\mathbf{x})\right]^{-1} \mathbf{f}(\mathbf{x}),$$

it is now straightforward to see that the situation is not different from the scalar case. We form an iterative problem: iteratively solve for $\mathbf{x}^{(k+1)}$, for $k = 0, 1, 2, \ldots$, starting with some initial guess $\mathbf{x}^0$. This can be done by solving for the update $\mathbf{H}^k$, using

$$J(\mathbf{x}^k)\mathbf{H}^k = -\mathbf{f}(\mathbf{x}^k),$$

which is a linear-algebraic problem, with an update

$$\mathbf{x}^{(k+1)} = \mathbf{x}^k + \mathbf{H}^{(k)}.$$

Convergence is now measured in terms of vector norms and the conditions for success or failure are more complex, but more geometrical (which is neat!). 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 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.

12.3.2 Iterative Techniques for Solving Linear Systems

We'll consider here three closely associated elementary methods: Jacobi (12.3.2), Gauss-Seidel (12.3.2), and Successive over-relaxation (SOR) (12.3.2). We will then discuss briefly Conjugate Gradient (for the solution of linear systems)(see 12.3.4), and later, the Multigrid method[7].

We want to solve

$$A\mathbf{x} = \mathbf{b},$$

here $\mathbf{x}$ and $\mathbf{b}$ are in $\mathbb{R}^n$, and A is an $n \times n$ matrix. The basic idea in Jacobi, Gauss-Seidel and SOR Solve using an initial guess $\mathbf{x}^0$ to $\mathbf{x}$, by generating a sequence $\left\{\mathbf{x}^{(k)}\right\}_{h=0}^{\infty}$ that converges to $\mathbf{x}$. This technique is especially well suited for large matrix problems, and is quite fast if the problem is well-conditioned. Moreover, it can be made very efficient if A has lots of zeros.

How?

Convert (12.3.2) into

(67)
$$\begin{aligned} \mathbf{x} &= T\mathbf{x} + \mathbf{c} \\ &\quad \text{after } \mathbf{x}^{(\mathbf{0})} \text{ is selected, then} \\ \mathbf{x}^{k+1} &= T\mathbf{x}^k + \mathbf{c} \quad k = 0, 1 \ldots \end{aligned}$$

Example Take

$$\begin{pmatrix} -1 & 0.1 & 0.5 & : & -0.3 \\ 0.2 & 10 & 2 & : & 22.2 \\ 0.5 & 1 & -3 & : & -0.5 \end{pmatrix}$$

has solution

$$\mathbf{x} = [1, 2, 1]^T$$

To convert (12.3.2)into $\mathbf{x} = T\mathbf{x} + \mathbf{c}$; solve for $x_i \quad i = 1, 2, 3$:

$$\begin{aligned} x_1 &= \frac{1}{10}x_2 + \frac{1}{2}x_3 + frac310 \\ x_2 &= -\frac{1}{50}x_1 - \frac{1}{5}x_3 + \frac{222}{100} \\ x_3 &= \frac{1}{6}x_1 + \frac{1}{3}x_2 + \frac{1}{6} \end{aligned}$$

[7]LINK multigrid

$$\text{Then } T = \begin{bmatrix} 0 & \frac{1}{10} & \frac{1}{2} \\ -\frac{1}{50} & 0 & -\frac{1}{5} \\ \frac{1}{6} & \frac{1}{3} & 0 \end{bmatrix} \mathbf{c} = \begin{bmatrix} \frac{3}{10} \\ \frac{222}{100} \\ \frac{1}{6} \end{bmatrix}$$

Take $\mathbf{x}^{(0)} = [0,0,0]^T$ as the initial guess. Solving above generates $\mathbf{x}^{(1)} = [0.9000, 2.0000, 0.6667]^T$. Then use this as a next guess and find $\mathbf{x}^{(2)}$, and so on. You'll find that you need about 10 iterations to get $\mathbf{x}^{10} = [1.0000, 2.0000, 1.0000]^T$.

In fact,

$$\frac{||x^{10} - x^9||_\infty}{||x^{10}||_\infty} = \frac{8 \cdot 0 \cdot 10^{-4}}{1.9998} < 10^{-3} \text{ in fact } ||\mathbf{x}^{10} - \mathbf{x}||_\infty = 0.0002.$$

□ This is the basic idea behind Jacobi iteration. So let's formalize it:

Jacobi Iteration

$$\text{Solve } x_i = \sum_{\substack{j=1 \\ i \neq j}}^{n} \left(-\frac{a_{ij} x_j}{a_{ii}} \right) + \frac{b_i}{a_{ii}} \quad i = 1, 2 \cdots n \tag{68}$$

To generate a sequence: for $k \geq 1 \ldots$

$$x_i^{(k)} = \frac{1}{a_{ii}} \left\{ \sum_{\substack{j=1 \\ i \neq j}}^{n} \left[1 - a_{ij} x_j^{(k-1)} + b_i \right] \right\} \quad i = 1, 2, \ldots, n$$

Compactly:

$$A = D - L - U =$$

$$= \begin{bmatrix} a_{11} & & & 0 \\ & a_{22} & & \\ & & \ddots & \\ 0 & & & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & \cdots & 0 & \\ -a_{21} & 0 & & \\ \vdots & & \ddots & \\ -a_{n1} & \cdots & -a_{n,n-1} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1,n} \\ & \vdots & & \\ & & 0 & -a_{n-1,n} \\ 0 & & & 0 \end{bmatrix}$$

$$\begin{array}{rc} \text{then } A\mathbf{x} = \mathbf{b} & \rightarrow D\mathbf{x} = (L+U)\mathbf{x} + \mathbf{b} \\ \mathbf{x} = & D^{-1}(L+U)\mathbf{x} + D^{-1}\mathbf{b} \end{array}$$

$$\boxed{\therefore \mathbf{x}^{k+1} = D^{-1}(L+U)\mathbf{x}^k + D^{-1}\mathbf{b}} \quad k = 0, 1 \ldots$$

Jacobi Algorithm

input: $n, \quad a_{ij} \quad , b_i \quad ; \quad X_i \quad ,$ TOL, NMAX (maximum number of iterations)

output: approximation of $\mathbf{x}$ or failure.

Step 1 $k = 1$

Step 2 While $(k \leq NMAX)$ do $3 - 6$

Step 3 for $i = 1 \ldots n$

$$x_i = -\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} X_j + b_i / a_{ii}$$

Step 4 if $||\mathbf{x} - \mathbf{X}|| <$ TOL then output $(x_1 \ldots x_n)$ STOP

Step 5 $k = k + 1$

Step 6 for $i = 1 \ldots n$ set $X_i = x_i$

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.

Can use $\dfrac{||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}||}{||\mathbf{x}^{(k)}||} <$ TOL in step 4 for stopping, also, might want to also examine $||\mathbf{b} - A\mathbf{x}^{(k)}||$. Usually the sup-norm is used.

Gauss-Seidel Method

A variant of Jacobi:

Look at (68) and use

$$x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}(a_{ij}x_j^{(k)}) - \sum_{j=i+1}^{n}(a_{ij}x_j^{k-1}) + b_i}{a_{ii}}$$

for each $i = 1, 2, \ldots, n$.

So, since

$x_i^{(k)}$ are to to be computed and these use $\mathbf{x}^{(k-1)}$. why not use

$x_1^{(k)} \cdots x_{i-1}^{(k)}$ which are likely better approximation than
$x_i^{(k-1)} \cdots x_{i-1}^{(k-1)}$, so use these the (k) values whenever these become available.

Compactly, Gauss-Seidel is $(D-L)\mathbf{x}^{(k)} = U\mathbf{x}^{(k-1)} + \mathbf{b}$

For $D-L$ to be non-singular it is necessary and sufficient that $a_{ii} \neq 0$.

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.

General Theory of Iterative Methods

Let's study methods of the form

$$\mathbf{x}^{(k)} = T\mathbf{x}^{k-1} + \mathbf{c}, \quad k = 1, 2, \cdots$$
$$x^{(0)} \text{the initial guess.}$$

To study convergence we recall the following fact:

Lemma If $\rho(T) < 1 \Rightarrow (I-T)^{-1}$ exists and

$$(I-T)^{-1} = I + T + T^2 + \cdots \tag{69}$$

Proof: if λ 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

$$T^m\mathbf{x} = \lambda^m\mathbf{x}$$

(here, $\mathbf{x} \neq 0$) 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 Let T as above and $\|T\| < 1$, then $(I-T)^{-1}$ exists and has the geometric series expansion (69). Moreover,

$$\|(I-T)^{-1}\| \leq \frac{1}{1-\|T\|}.$$

Proof: use the above 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 $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ defined by

$$\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}, \quad k \geq 1 \quad c \neq 0$$

converges to the unique solution of

$$\mathbf{x} = T\mathbf{x} + \mathbf{c}$$
$$\text{if and only if } \rho(T) < 1.$$

Proof: Use induction to get $\mathbf{x}^{(k)} = T^k\mathbf{x}^{(0)} + (T^{k-1}\cdots T + I)\mathbf{c}$.

$$\text{If } \rho(T) < 1 \Rightarrow \lim_{k\to\infty} T^k\mathbf{x}^{(0)} = 0.$$

Use previous Lemma and show that $\mathbf{x} - \mathbf{x}^{(k)} = 0$ as $k \to \infty$. □

Corollary If $||T|| < 1$ in some induced norm $\Rightarrow \left\{x^{(k)}\right\}$ converges for any $\mathbf{x}^{(0)} \in \mathbb{R}^n$ to $\mathbf{x} \in \mathbb{R}^n$ and

(i) $||\mathbf{x} - \mathbf{x}^{(k)}|| \leq ||T||^k ||\mathbf{x}^0 - \mathbf{x}||$

(ii) $||\mathbf{x} - \mathbf{x}^{(k)}|| \leq \frac{||T||^k}{1-||T||} ||\mathbf{x}^1 - \mathbf{x}^{(0)}||$

□

Theorem: If A is strictly diagonal dominant then for any $x^{(0)}$ Jacobi and Gauss-Seidel converge to the unique solution of $A\mathbf{x} = \mathbf{b}$

Remark: Which method to pick? Since $||\mathbf{x}^{(k)} - \mathbf{x}|| \approx \rho(T)^k ||\mathbf{x}^{(0)} - \mathbf{x}||$ 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 1 and only 1 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) $\rho(T_J) = \rho(T_G) = 1$

where T_J is the matrix resulting from Jacobi and T_G the matrix resulting from Gauss-Seidel.

□

Definition: let $\tilde{\mathbf{x}} \in \mathbb{R}^n$ be approximation to $\mathbf{x}$ of solution to $A\mathbf{x} = \mathbf{b}$. Let $\mathbf{r} = \mathbf{b} - A\mathbf{x}$.

In Gauss-Seidel $\mathbf{x}_i^{(k)} = (\mathbf{x}_1^{(k)}, \mathbf{x}_2^{(k)} \cdots \mathbf{x}_{i-1}^{(k)}, \mathbf{x}_i^{(k-1)}, \mathbf{x}_{i+1}^{(k-1)}, \ldots, \mathbf{x}_n^{(k-1)})^T$

define: $\mathbf{r}_i^{(k)} = (\mathbf{r}_{1i}^{(k)}, \mathbf{r}_{2i}^{(k)}, \ldots \mathbf{r}_{ni}^{(k)})^T$.

The m^{th} component of $\mathbf{r}_1^{(k)}$'s

$$r_{mi}^{(k)} = b_m - \sum_{j=1}^{i-1} a_{mj} x_j^{(k)} - \sum_{j=i+1}^{n} a_{mj} x_j^{(k-1)} - a_{mi} x_i^{(k-1)}, \tag{70}$$

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

$$a_{ii} x_i^{(k-1)} + r_{ii}^{(k)} = b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \tag{71}$$

but Gauss-Seidel:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right] \tag{72}$$

So (12.3.2) can be written as

$$a_{ii} x_i^{(k-1)} + r_{ii}^{(k)} = a_{ii} x_i^{(k)}$$

therefore Gauss-Seidel can be characterized as choosing $x_i^{(k)}$ to satisfy

$$x_i^{(k)} = x_i^{(k-1)} + \frac{r_{ii}^{(k)}}{a_{ii}}, \tag{73}$$

wherein we want to reduce the size of r_{ii} to zero. Another connection: Consider $\mathbf{r}_{i+1}^k$ associated with

$\mathbf{x}_{i+1}^{(k)} = (x_1^{(h)}, \ldots x_i^{(k)}, x_{i+1}^{(k-1)}, \ldots, x_n^{(k-1)})^T$. By (70)

$$\begin{aligned} r_{i,i+1}^{(k)} &= b_i - \sum_{j=1}^{i} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \\ &= b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} - a_{ii} x_i^{(k)} \end{aligned}$$

Equation (72) implies that $r_{i,i+1}^{(k)} = 0$. In this sense Gauss-Seidel is also characterized by requiring that the i^{th} component of $r_{i+1}^{(k)}$ be zero.

Reducing one coordinate of the residual to zero is not generally the most efficient way to reduce the norm of $r_{i+1}^{(k)}$. Instead:

Modify (72):

$$x_i^{(k)} = x_i^{(k-1)} + \omega \frac{r_{ii}^{(k)}}{a_{ii}} \quad \text{"Relaxation Method"}$$
$$0 < \omega$$

For $0 < \omega < 1$ we get "under-relaxation". Can use to get convergence for some choices where Gauss-Seidel fails to converge.

$1 < \omega$ we get "over-relaxation". Use to obtain faster convergence when Gauss-Seidel converges. So this can be codified in another method:

Successive Over Relaxation (SOR)

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}}[b_i - \sum_j^{i-1} a_{ij}x_j^{(k)}] = (1-\omega)a_{ii}x_i^{k-1} - \omega \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} + \omega b_i,$$

Or more compactly,

$$\mathbf{x}_k = \underbrace{(D-\omega L)^{-1}[(1-\omega)D + \omega U]}_{T_\omega}\mathbf{x}^{k-1} + \omega \mathbf{b}.$$

How do we choose ω? 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 positive definite and $O < \omega < 2$ then SOR converges for any $\mathbf{x}^{(0)}$.

□

Theorem: If A positive definite tridiagonal then $\rho(T_G) = [\rho(T_J)]^2 < 1$. Then, the optimal choice ω for SOR is

$$\omega = \frac{2}{1 + \sqrt{1 - \rho(T_G)}} = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}}.$$

Then $\rho(T_\omega) = \omega - 1$

□

SOR Algorithm:

Input: Same as before and the parameter ω

Output: $\mathbf{x}$

Step 1 $k = 1$
Step 2 While $(k \leq NMAX)$ do 3 - 6
Step 3 for $i = 1, \ldots, n$

$$x_i = (1-\omega)X_i + \frac{\omega\left(-\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}X_j + b_i\right)}{a_{ii}}$$

Step 4 if $||\mathbf{x} - \mathbf{X}|| < \text{TOL}$, output $\mathbf{x}$, END
Step 5 $k = k + 1$
Step 6 for $i = 1, \ldots, n \qquad X_i = x_i$
Step 7 FAIL MESSAGE, STOP.
END

□

Another look at convergence of iterative methods:

We write Jacobi, Gauss-Seidel, SOR as

$$\mathbf{x}^{(n+1)} = T\mathbf{x}^{(n)} + \mathbf{c}$$

The error at the n^{th} step is $\mathbf{e}^{(n+1)} = \mathbf{x} - \mathbf{x}^{(\mathbf{n})}$ where $\mathbf{x}$ is the solution of $A\mathbf{x} = \mathbf{b}$.

Hence $\mathbf{e}^{(n+1)} = T\mathbf{e}^{(n)}$

Using inductions $\mathbf{e}^{(n)} = T^n\mathbf{e}^{(0)}$

The sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(\mathbf{2})}, \cdots, x^{(n)}, \cdots$ will converge to $\mathbf{x}$ as n tends to infinity is $\lim_{n\to\infty^n} e^{(n)} = \mathbf{0}$.

Assume T is an $m \times m$ matrix. Assume it has m linearly independent eigenvectors $\mathbf{v}_s; s = 1, 2 \cdots m$ and write

$$\mathbf{e}^{(\mathbf{0})} = \sum_{s=1}^{m} c_s\mathbf{v}_s \quad , \quad \text{where } c_s\text{are scalars. Hence,}$$

$$\mathbf{e}^{(1)} \;=\; T\mathbf{e}_n^0 = \sum_{s=1}^{m} c_s \lambda_s \mathbf{v}_s$$

where λs are the eigenvalues of T.

Similarly,

$$\mathbf{e}^{(n)} = \sum_{s=1}^{m} c_s \lambda_s^n \mathbf{v} s$$

Therefore $\mathbf{e}^{(n)}$ tends to the null vector as $n \to \infty$ for any $\mathbf{e}^{(n)}$ 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 m linear equations

$$A\mathbf{x} = \mathbf{b}$$

are such that the matrix $A = D - L - U$ is nonsingular and has nonzero diagonal elements $a_{ii} \quad i = 1, 2, \cdots, m$.

Jacobi Iteration

The eigenvalues μ of the Jacobi iteration $T \equiv D^{-1}(L + U)$ are roots of

$$\det[\mu I - D^{-1}(L+U)] \;=\; \det D^{-1}(\mu D - L - U)$$
$$= \det D^{-1} \det(\mu D - L - U) = 0$$

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 $T \equiv (I - WD^{-1}L)^{-1}\left\{(1-w)I + wD^{-1}U\right\}$
for which the eigenvalues λ are roots of $\det(\lambda I - T) = 0$.
Now $\lambda I - T = \lambda(I - WD^{-1}L)^{-1}(I - WD^{-1}L) - (I - WD^{-1}L)^{-1}\left\{(1-w)I + wD^{-1}U\right\}$
$= (I - WD^{-1}L)^{-1}\left\{\lambda(I - wD^{-1}L) - (1-w)I - wD^{-1}U\right\}$
$= (I - wD^{-1}L)^{-1}D^{-1}\left\{(\lambda + w - 1)D - \lambda wL - wU\right\}$

Therefore

$$\begin{aligned} \det(\lambda I - T) &= \det\left\{(\lambda + w^{-1})0 - \lambda w L - wU\right\} / \det(I - wD^{-1}L) \det D \\ \text{but } \det(I - WD^{-1L}) &= \text{determinant of a unit lower triangular matrix } = 1. \\ \text{Also } \det D &= a_{11}a_{22}\cdots a_{mm} \neq 0 \end{aligned}$$

Hence the eigenvalues λ are roots of

$$\det\{(\lambda + w - 1)D = \lambda w L - wU\} = 0$$

This result holds for any set of linear equations satisfying $\det A \neq 0$ and $\det D \neq 0$. The eigenvalues λ of the Gauss-Seidel are given by setting $w =$ (i.e. they are the roots λ of $\det(D - \lambda L - U) = 0$.

□

Iterative Refinement One would think that if $\tilde{\mathbf{x}}$ is approximate to $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$ and $\mathbf{r} = \mathbf{b} - A\mathbf{x}$, then if $||\mathbf{r}||$ is small then $||\mathbf{x} - \tilde{\mathbf{x}}||$ is small. However, this is not always the case. Consider

Example

$$\left[\begin{array}{lcl} 1 & 2 & 3 \\ 1.0001 & 2 & 3.0001 \end{array}\right].$$

It has a unique solution $\mathbf{x} = (1, 1)^T$. Now, take $\tilde{x} \equiv (3, 0)^T$. We compute $||\mathbf{r}||_\infty = 0.0002$. Although small, $\tilde{\mathbf{x}}$ is a poor approximation to $\mathbf{x}$, i.e. $||\mathbf{x} - \tilde{\mathbf{x}}||_\infty = 2$.

□

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 $\frac{||\mathbf{x}-\mathbf{x}||}{||\mathbf{x}||} \leq \kappa(A)\frac{||\mathbf{r}||}{||\mathbf{b}||}$, where $\kappa(A)$ is the condition number for the matrix A.

□

So, using the above example, $||A||_\infty = 3.001$, and

$$A^{-1} = \left[\begin{array}{ll} -10000 & 10000 \\ 5000.5 & -5000 \end{array}\right]$$

$||A^{-1}||_\infty = 20000$ 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 $\mathbf{b}(\mathbf{x} - \tilde{\mathbf{x}}) = A\mathbf{x}A^{-1}\mathbf{r}$.

□

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 $\mathbf{r} = \mathbf{b} - A\mathbf{x}$.

where $\mathbf{x} \approx \tilde{\mathbf{x}} + \tilde{\mathbf{y}}$ till we get a better $\mathbf{y}$.
Then solve for $\tilde{\mathbf{x}}$

□

12.3.3 Ill-conditioning and Finite Precision Errors

So far we've assumed $A, \mathbf{x}, \mathbf{b}$ are represented exactly on a machine. However, what we really might have on a machine is

$$(A + \delta A)\mathbf{x} = \mathbf{b} + \delta\mathbf{b}, \quad \text{instead of } A\mathbf{x} = \mathbf{b}.$$

If $||\delta A||$ and $||\delta\mathbf{b}||$ 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.

Suppose A is ill-conditioned. In this case, no matter how small these perturbations to A and $\mathbf{b}$ are, the errors will propagate, sometimes to disastrous proportions, especially if the size of the matrix is large:

Theorem: If A is nonsingular and $||\delta A|| < \frac{1}{||A^{-1}||}$ then the solution $\tilde{\mathbf{x}}$ to $(A + \delta A)\tilde{\mathbf{x}} = b + \delta\mathbf{b}$ approximates $A\mathbf{x} = \mathbf{b}$ with error

$$\frac{||\tilde{\mathbf{x}} - \mathbf{x}||}{||\mathbf{x}||} \leq \frac{\kappa(A)}{1 - \kappa(A)(||\delta A||/||A||)} \left(\frac{||\delta\mathbf{b}||}{||\mathbf{b}||} + \frac{||\delta A||}{||A||} \right).$$

Therefore, if A is well-conditioned $\kappa(A)$ is small and perturbations in A and $\mathbf{b}$ produce small changes in $\mathbf{x}$ and vice versa.

□

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

$$(A + \delta A)\mathbf{x} = \mathbf{b}$$

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 $f(n) = 1.01(n^3 + 3n^2)$.

□

12.3.4 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 a "Steepest Descent" method, aimed at solving linear algebraic problems. 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. This means that we search for the vector for which the gradient of the functional is zero, and this can be done in an iterative fashion. A special steepest descent method, appropriate for the solution of the linear algebraic problem is called the "Conjugate Gradient" Method. We feature it next.

We want to solve

$$A\mathbf{x} = \mathbf{b} \quad \mathbf{x} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}. \tag{74}$$

Suppose A is symmetric positive definite $\begin{cases} A^T = A \\ \mathbf{x}^T A\mathbf{x} > 0, \quad x \neq 0. \end{cases}$

Lemma: If A is symmetric positive definite then solving $A\mathbf{x} = \mathbf{b}$ is equivalent

to problem of minimizing the quadratic form

$$\mathbf{f}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$$

where $f \in \mathbb{R}^n$. The minimum is reached when $\mathbf{x} = A^{-1}\mathbf{b}$.

□

$\mathbf{f}$ is differentiable, and in fact

$$-\nabla\mathbf{f}(\mathbf{x}) = -A\mathbf{x} + \mathbf{b}.$$

So looking for an extrema of $\mathbf{f}$ means solving for $-\nabla\mathbf{f} = \mathbf{0}$. Moreover, $\nabla^2\mathbf{f} = A$, and since $\|A\| > 0$ then the extrema is a minima.

Hence, we recast the problem of finding a solution of the (74) as the optimization problem: find $\hat{\mathbf{x}}$ such that

$$\mathbf{f}(\hat{\mathbf{x}}) = \min_{\mathbf{x}\in\mathbb{R}^n} \mathbf{f}(\mathbf{x}),$$

which is equivalent to finding $\hat{\mathbf{x}}$ the solution of (74). Next we observe that

$$\mathbf{f}(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T A(\mathbf{x} - \hat{\mathbf{x}}) - \frac{1}{2}\hat{\mathbf{x}}^T A\hat{\mathbf{x}}$$

and minimization is the same as that of

$$\hat{\mathbf{f}} \equiv \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T A(\mathbf{x} - \hat{\mathbf{x}}) \tag{75}$$

since $\mathbf{f}$ and $\hat{\mathbf{f}}$ differ by a constant independent of $\mathbf{x}$. But minimizing $\hat{\mathbf{f}}$ is nonsensical since it involves $\hat{\mathbf{x}}$ the unknown in the computation.

However, since

$$A(\mathbf{x} - \hat{\mathbf{x}}) = (A\mathbf{x} - \mathbf{b}) - (A\hat{\mathbf{x}} - \mathbf{b}) = \mathbf{r}$$

where $\mathbf{r}(\mathbf{x}) \equiv A\mathbf{x} - \mathbf{b}$. Note that $\mathbf{r}(\hat{\mathbf{x}}) = 0$. Thus

$$\mathbf{f} = \frac{1}{2}\mathbf{r}^T A\mathbf{r}$$

and it is clear then that

$$-\nabla\mathbf{f} = \mathbf{b} - A\mathbf{x} = \mathbf{r}.$$

Next, we recast the minimization as an iterative map: use k as the iteration counter and let

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_{k+1}\mathbf{p}_{k+1}$$

which is called a *line search*, along the direction $\mathbf{p}_{k+1}$. Here α is the *search parameter* and is used to minimize the functional $\mathbf{f}(\mathbf{x}^{(k)}+\alpha_{k+1}\mathbf{p}_{k+1})$ along the direction $\mathbf{p}_{k+1}$. This would be somewhat daunting in many space directions, however, we can find α analytically: the minimum α_{k+1} occurs when the new residual is orthogonal to the search direction:

$$\begin{aligned} 0 &= \frac{d}{d\alpha_{k+1}}\mathbf{f}(\mathbf{x}^{(k+1)}) = \nabla\mathbf{f}(\mathbf{x}^{(k+1)})^T\frac{d}{d\alpha_{k+1}}\mathbf{x}^{(k+1)} \\ &= (A\mathbf{x}_{k+1}-\mathbf{b})^T(\frac{d}{d\alpha_{k+1}}(\mathbf{x}^{(k)}+\alpha_{k+1}\mathbf{p}_{k+1})) = -\mathbf{r}_{k+1}^T\mathbf{p}_{k+1}. \end{aligned}$$

The new residual can be expressed in terms of the old residual:

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{b} - A\mathbf{x}_{k+1} \\ &= \mathbf{b} - A(\mathbf{x}^{(k)} + \alpha_{k+1}\mathbf{p}_{k+1}) \\ &= \mathbf{b} - A\mathbf{x}_k - \alpha_{k+1}A\mathbf{p}_{k+1} \\ &= \mathbf{r}_k - \alpha_{k+1}A\mathbf{p}_{k+1}. \end{aligned}$$

Thus, solving,

$$\alpha_{k+1} = \frac{\mathbf{r}_k^T\mathbf{r}}{\mathbf{p}_{k+1}^TA\mathbf{p}_{k+1}}.$$

So we know how to get α and we can find $\mathbf{p}$ by the requirement that it has to be orthogonal to $\mathbf{r}$. But how do we find $\mathbf{p}_{k+1}$ at each iterate k?

To proceed, we need to make the following definition:

A-Conjugate vectors:

A set of non-zero vectors $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n \in \mathbb{R}^n$ is said to be A-conjugate if

$$\delta_{ij} = \mathbf{p}_i^TA\mathbf{p}_j, \quad 1 \le i, j \le n. \tag{76}$$

The $\mathbf{p}_j$ are called "conjugate directions." Condition (76) is equivalent to requiring that $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ 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, (75) is no more than the norm $\frac{1}{2}||\cdot||_A^2$, i.e.

$$\tilde{f}(x) = \frac{1}{2}\|\mathbf{x} - \tilde{\mathbf{x}}\|_A^2.$$

Take

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{p}_k, \tag{77}$$

$k = 0, 1, \ldots$. Suppose we start (77) with $\mathbf{x}^{(0)} = \mathbf{0}$. We can always do this: to see this, let $A\mathbf{z} = \mathbf{b} - A\mathbf{x}^{(0)}$ with solution $\mathbf{z}^*$, then $\mathbf{x}^* = \mathbf{x}_0 + \mathbf{z}^*$ hence an initial guess $\mathbf{z}^{(0)} = \mathbf{0}$ corresponds to $\mathbf{x}^{(0)} = \mathbf{x}_0$ in the original problem.

Express

$$\tilde{\mathbf{x}}^{(k)} = \sum_{j=1}^{n} \alpha_j \mathbf{p}_j,$$

there solution of $A\mathbf{x} = \mathbf{b}$. Multiply both sides of this equation by A and then by $\mathbf{p}_{k+1}$, then using orthogonality

$$\alpha_{k+1} = \frac{\mathbf{p}_{k+1}^T A\tilde{\mathbf{x}}}{\mathbf{p}_{k+1}^T A\mathbf{p}_{k+1}} = \frac{\mathbf{p}_{k+1}^T A\mathbf{b}}{\mathbf{p}_{k+1}^T A\mathbf{p}_{k+1}}, \qquad k = 0, 1, 2, \ldots n-1.$$

Note that when $k = n$, $\mathbf{x}_n = \tilde{\mathbf{x}}$ and $\mathbf{r}_n = 0$. 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 $\{\mathbf{p}_k\}$ and $\{\mathbf{x}_k\}$: Note that $\mathbf{r}_0 = \mathbf{b}$, if $\mathbf{x}^0 = 0$, and that $\mathbf{r}_k$ is orthogonal to $\mathbf{p}_i$, $i = 1, 2, \ldots, n$, i.e.

$$\mathbf{r}_k^T A\mathbf{p}_i = \delta_{k,i}.$$

Starting with $\mathbf{p}_1 = \mathbf{b}$ the conjugate gradient is then:

$$\begin{array}{ll}
\alpha_{k+1} = \mathbf{r}_k^T\mathbf{r}_k / \mathbf{p}_{k+1}^T A\mathbf{p}_{k+1} & k \geq 0 \\
\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_{k+1}\mathbf{p}_{k+1} & k \geq 0 \\
\beta_{n+1} = \mathbf{r}_k^T\mathbf{r}_k / \mathbf{r}_{k-1}^T\mathbf{r}_{k-1} & k > 0 \\
\mathbf{p}_{k+1} = \mathbf{r}_k + \beta_{k+1}\mathbf{p}_k & k > 0 \\
\mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}^{(k+1)} & k \geq 0
\end{array}$$

The iterates converge to solution of $A\mathbf{x} = \mathbf{b}$ in at most n iterations!

□

Example Solve $A\mathbf{x} = \mathbf{b}$ with

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

by the conjugate gradient method.

Let $\mathbf{x}^{(0)} = \mathbf{0}$, then $\mathbf{r}_0 = \mathbf{p}_1 = \mathbf{b} = [1\,0\,1]^T$.

Form $\mathbf{r}_0^T\mathbf{r}_0 = 2$ so

$$\mathbf{p}_1^T A\mathbf{p}_1 = [1,\,0\,1][a_{ij}] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 4$$

Therefore $\alpha_1 = \mathbf{r}_0^T\mathbf{r}_0/\mathbf{p}_1^T A\mathbf{p}_1 = 0.5$ and

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha_1\mathbf{p}_1 = \begin{bmatrix} 0.5 \\ 0.0 \\ 0.5 \end{bmatrix}$$

For the next iteration $\mathbf{r}_1 = \mathbf{b} - A\mathbf{x}^{(1)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

$$\beta_2 = \mathbf{r}_1^T\mathbf{r}_1/\mathbf{r}_0^T\mathbf{r}_0 = \frac{1}{2} = 0.5 \therefore$$

$$\mathbf{p}_2 = \mathbf{r}_1 + \beta_2\mathbf{p}_1 = [0.5\,1.0\,0.5]^T$$

$$\therefore \mathbf{p}_2^T A\mathbf{p}_2 = 1 \quad \Rightarrow \alpha_2 = 1 \quad \text{and}$$

$$\mathbf{x}^{(2)} = \mathbf{x}^{(1)} + \alpha_2\mathbf{p}_2 = [1\,1\,1]^T$$

the residual $\mathbf{r}_2 = \mathbf{0}$ and $\mathbf{x}^{(2)}$ is the solution. □

A conservative estimate of the rate of convergence of the conjugate gradient method is

$$\|\mathbf{x} - \tilde{\mathbf{x}}\|_A \leq 2\left[\frac{1-\sqrt{c}}{1+\sqrt{c}}\right]\|\tilde{\mathbf{x}}\|_A.$$

Here, $c = \lambda_1/\lambda_n$, where $0 < \lambda_1 \leq \lambda_2 ... \leq \lambda_n$ 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 ill-conditioned A, the Conjugate Gradient Method is very slow.

What to do?

Remark: For ill-conditioned A, the Conjugate Gradient Method is very slow.

What to do?

Preconditioning Take

$$A\mathbf{x} = \mathbf{b}$$

and it turn into

$$B\mathbf{y} = \mathbf{d}.$$

where

$$B = Q^T AQ \quad , \quad \mathbf{y} = Q^{-1}\mathbf{x} \quad \text{and } \mathbf{d} = Q^T\mathbf{b}.$$

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 $B\mathbf{y} = \mathbf{d}$ is actually not computed or used. There is a modification that allows us to use the above discussed conjugate gradient method (see reference to Gollub and van Loan.

□

Comparison of Convergence of Jacobi 12.3.2, Gauss-Seidel 12.3.2, SOR 12.3.2, and Conjugate Gradient 12.3.4. Let's do it in the context of solving the linear-algebraic problem related to the "Poisson Problem" for $v(x, y)$ which is considered in detail under Elliptic Problems

$$\Delta v = f(x, y)$$

over some domain in the 2D-plane x, y, with suitable boundary conditions at the edges of the domain using the 3 (in 1D) or 5 point stencil. Using the

3 or 5 point stencil and finite-difference techniques, it is possible to turn the problem of solving approximately the above partial differential equation into a linear-algebraic equation

$$A\mathbf{z} = \mathbf{b}$$

where $\mathbf{z}$ 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.

So to solve the "Poisson Problem" amounts to solving the linear algebraic problem stated above.

Define the spectral radius of the Jacobi case as $\rho(A_J)$ (see 11.2).

$$\rho(A_J) \approx 1 - \frac{a}{n} \quad \begin{array}{l} \text{as } n(\text{ number of unknowns}) \to \text{grows } \rho \to 1. \\ \text{Since the error is multiplied by} \\ \rho \Rightarrow \text{convergence slows down.} \end{array}$$

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 $\rho(A_{SOR}) \approx 1 - \dfrac{a'}{\sqrt{n}}$ 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 $\kappa = \mathcal{O}(n)$. Hence, after k steps in the Conjugate Gradient Method the residual is multiplied by about $[1-\mathcal{O}(\sqrt{n})]^k$, same as optimal SOR, thus the CG takes about $\mathcal{O}(\sqrt{n})$.

□

13 NUMERICAL TECHNIQUES FOR EIGENVALUES

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, and inverse iteration, QR. Another classic method that's enjoying a revival is Arnoldi Iteration, used in the estimation of eigenvalues of very large matrices.

13.1 The Power Method

Used to estimate the dominant eigenvalue and its corresponding eigenvector.

There's two variants: the case when A ($n \times n$ matrix) has

$$\begin{cases} n \text{ linearly independent eigenvectors} \\ \text{there's a single dominant eigenvalue.} \end{cases}$$

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.

The first case: first, we order the eigenvalues

(i) $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$

(ii) There's a basis $\left\{u^{(1)}, u^{(2)}, \cdots, u^{(n)}\right\}$ for $\mathbb{C}^n$ such that

$$Au^{(i)} = \lambda_i u^{(i)} \quad (1 \leq i \leq n) \tag{78}$$

Let $x^{(0)}$ be any element of $\mathbb{C}^n$ such that when $x^{(0)}$ is expressed as a linear combination of the basis $u^{(1)}, u^{(2)}, \ldots, u^{(n)}4$ the coefficient of $u^{(1)}$ is not 0. Thus,

$$x^{(0)} = a_1 u^{(1)} + a_2 u^{(2)} + \cdots u_n u^{(n)} \quad a_1 \neq 0 \tag{79}$$

We form

$$\begin{aligned} x^{(1)} &= Ax^{(0)}, \\ x^{(2)} &= Ax^{(1)}, \\ &\cdots \\ x^{(k)} &= Ax^{(k-1)} \end{aligned}$$

so that

$$x^{(k)} = A^k x^{(0)} \tag{80}$$

In what follows we'll absorb all the coefficients a_j in the vectors $u^{(j)}$ that they multiply. Hence, write (79) as

$$x^{(0)} = u^{(1)} + u^{(2)} + \ldots + u^{(n)} \tag{81}$$

by (81) we can write (80) as

$$x^{(k)} = A^{(k)}u^{(1)} + A^{(k)}u^{(2)} + \ldots A^k u^{(n)}$$

using (78)

$$x^{(k)} = \lambda_1^k u^{(1)} + \lambda_2^k u^{(2)} + \ldots \lambda_n^k u^{(n)}$$

rewrite this

$$x^{(k)} = \lambda_1^k \left[u^{(1)} + \left(\frac{\lambda_2}{\lambda_1}^k\right) u^{(2)} + \ldots \left(\frac{\lambda_n}{\lambda_1}\right)^k u^{(n)} \right] \tag{82}$$

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 (82) 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 θ 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 α, β are scalars and x and y vectors. Then

$$\theta(x^{(k)}) = \lambda_1^k [\theta(u^{(1)}) + \theta(\varepsilon^{(k)})]$$

$\therefore$ the following ratios converge to λ_1 as $k \to \infty$:

$$r_k \equiv \frac{\theta(x^{(k+1)})}{\theta(x^{(k)})} = \lambda_1 \left[\frac{\theta(u)^{(1)} + \theta(\varepsilon^{(k+1)})}{\theta(u^{(1)}) + \theta(\varepsilon^{(k)})} \right] \to \lambda_1$$

□

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 θ? It can 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.

The algorithm goes like this:

input: n, A, x, m % here x is an initial guess

output: $0, x$

$$\begin{aligned} \text{do } k &= 1, m \\ y &= Ax \\ r &= \theta(y)/\theta(x) \\ x &= y/||y|| \quad \% \text{ use sup-norm for example.} \end{aligned}$$

end

Relative error: 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:

$$\begin{aligned} r_{k+1} - \lambda_1 &= (c + \delta k)(r_k - \lambda_1) \\ |c| < 1 \text{ and } &\lim_{k\to\infty} \delta_k = 0 \end{aligned}$$

this implies 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 $n \to \infty$.

□

13.2 Inverse Power Method

We know that if λ is an eigenvalue of A and if A is nonsingular then λ^{-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

$$|\lambda_1| \geq |\lambda_2| \geq \ldots \leq |\lambda_{n-1}| > |\lambda_n| > 0$$

can be done since A is non-singular, 0 is not an eigenvalue. The eigenvalues of A^{-1} arranged:

$$|\lambda_n^{-1}| > |\lambda_{n-1}^{-1}| \geq \cdots \geq |\lambda_1^{-1_1} > 0$$

$\therefore$ Apply power method to A^{-1}. But we don't compute A^{-1}. Instead 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 suggest a way to find the eigenvalue farthest to a given value μ. The "Shifted Matrix Power Method," here μ 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 μ. In this case we apply the inverse power method on $\hat{A}$, i.e.

$$\hat{A}x^{(k+1)} = x^{(k)}$$

□

13.3 The Rayleigh-Ritz Method:

If A is symmetric one can use an old, but efficient method. Note that eigenvalues and eigenvectors of A are real.

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 ever more significant. □

13.3.1 Rayleigh-Ritz, Background:

Let's consider more generally the case for A an $n\times$ Hermitian matrix and $\mathbf{x}$ is an $n-$dimensional vector. We indicate "hermitian" as †, which means that the complex conjugate of the transpose of A is the same as A. So, for short, $A^\dagger = A$. If A is a real matrix, $A^T = A$. These matrices often arise from self-adjoint continuous operators which model some physical process. The complex version appears often in the context of quantum mechanics and acoustics.

We will indicate by an overbar the operation of taking the conjugate transpose. If A and $\mathbf{x}$ were real, this operation would simply involve the transpose.

Since $A^\dagger = A$ the eigenvalues are real and can be organized as

$$\lambda_{min} \equiv \lambda_1 \leq \lambda_2 \leq \lambda_3 \cdots \leq \lambda_n \equiv \lambda_{max}.$$

We will see that the smallest and largest eigenvalues may be thought of as the solution to a constrained minimum and maximum problem.

Theorem (Rayleigh-Ritz): Let A as above and the eigenvalues ordered as above. Then

$$\lambda_1 \overline{\mathbf{x}}\mathbf{x} \leq \overline{\mathbf{x}} A \mathbf{x} \leq \lambda_n \overline{\mathbf{x}}\mathbf{x}, \qquad \text{for all} \mathbf{x} \in C^n.$$

Furthermore,

$$\lambda_{max} = \max_{\mathbf{x}\neq 0} \frac{\overline{\mathbf{x}} A \mathbf{x}}{\overline{\mathbf{x}}\mathbf{x}} = \max_{|\mathbf{x}|^2=1},$$

and

$$\lambda_{min} = \min_{\mathbf{x}\neq 0} \frac{\overline{\mathbf{x}} A \mathbf{x}}{\overline{\mathbf{x}}\mathbf{x}} = \min_{|\mathbf{x}|^2=1}.$$

Proof: Since $A = A^\dagger$ then there exists a unitary matrix U such that $A = U\Lambda U^\dagger$, with $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$. For any $\mathbf{x} \in C^n$ we have

$$\overline{\mathbf{x}}A\mathbf{x} = \overline{\mathbf{x}}U\Lambda U^\dagger\mathbf{x} = (U^\dagger\mathbf{x})^\dagger\Lambda(U^\dagger\mathbf{x}) = \sum_{i=1}^{n}\lambda_i|(U^\dagger\mathbf{x})_i|^2.$$

Since $|(U^\dagger\mathbf{x})_i|^2$ is non-negative, then

$$\lambda_{min}\sum_{i=1}^{n}\lambda_i|(U^\dagger\mathbf{x})_i|^2 \leq \overline{\mathbf{x}}A\mathbf{x} = \sum_{i=1}^{n}\lambda_i|(U^\dagger\mathbf{x})_i|^2 \leq \lambda_{max}\sum_{i=1}^{n}\lambda_i|(U^\dagger\mathbf{x})_i|^2.$$

Because U is unitary

$$\sum_{i=1}^{n}\lambda_i|(U^\dagger\mathbf{x})_i|^2 = \sum_{i=1}^{n}|\mathbf{x}_i|^2 = \overline{\mathbf{x}}\mathbf{x}.$$

Hence,

$$\lambda_1\overline{\mathbf{x}}\mathbf{x} = \lambda_{min}\overline{\mathbf{x}}\mathbf{x} \leq \overline{\mathbf{x}}A\mathbf{x} \leq \lambda_{max}\overline{\mathbf{x}}\mathbf{x} = \lambda_n\overline{\mathbf{x}}\mathbf{x}.$$

These are sharp inequalities. If $\mathbf{x}$ is an eigenvector of A associated with λ_1, then

$$\overline{\mathbf{x}}A\mathbf{x} = \lambda_1\overline{\mathbf{x}}\mathbf{x}.$$

Same sort of argument holds for λ_n.

Furthermore, if $\mathbf{x} \neq 0$ then

$$\frac{\overline{\mathbf{x}}A\mathbf{x}}{\overline{\mathbf{x}}\mathbf{x}} \leq \lambda_n.$$

so

$$\max_{\mathbf{x}\neq 0}\frac{\overline{\mathbf{x}}A\mathbf{x}}{\overline{\mathbf{x}}\mathbf{x}} = \lambda_n. \tag{83}$$

Finally, since $\mathbf{x} \neq 0$, then

$$\frac{\overline{\mathbf{x}}A\mathbf{x}}{\overline{\mathbf{x}}\mathbf{x}} = \frac{\overline{\mathbf{x}}}{\sqrt{\overline{\mathbf{x}}\mathbf{x}}}A\frac{\mathbf{x}}{\sqrt{\overline{\mathbf{x}}\mathbf{x}}}$$

and

$$\frac{\overline{\mathbf{x}}}{\sqrt{\overline{\mathbf{x}}\mathbf{x}}}\frac{\mathbf{x}}{\sqrt{\overline{\mathbf{x}}\mathbf{x}}} = 1.$$

Hence, (83) is equivalent to

$$\max_{|\mathbf{x}|^2=1} \overline{\mathbf{x}} A\mathbf{x} = \lambda_n.$$

Same sort of arguments hold for λ_1, in the context of the minimum. □

Algorithm

Now we will revert to the case of A an $n \times n$ symmetric real matrix for the presentation of the algorithm.

Let $\mathbf{x}$ be an $n-$dimensional real vector. Choose some initial guess $x^{(0)}$, and compute

$$x^{(m+1)} = Ax^{(m)}$$

then

$$\lambda_1^{(m+1)} = \frac{(x^{m+1}, x^{(m)})}{(x^{(m)}, x^{(m)})} \quad m \geq 0$$

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. □

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. Given A, there's a factorization

$$A = QR \quad ,$$

R is upper triangular and Q orthogonal. With A real both Q, R are chosen real. The motivation 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 $||w||_2 = \sqrt{w^\star w} = 1$. Define

$$U = I - 2ww^\star \tag{84}$$

this is the general form of a "Householder Matrix"

Example) For $n = 3$

$$w = [w, w_2, w_3]^T \quad |w_1|^2 + |w_2|^2 + |w_3|^2 > 1$$

$$U = \begin{bmatrix} 1 - 2|w_1|^2 & -2w_1\overline{w}_2 & -2w_1\overline{w}_3 \\ -2\overline{w}_1 w_2 & 1 - 2|w_2|^2 & -2w_2\overline{w}_3 \\ -2\overline{w}_1 w_3 & -2\overline{w}_2 w_3 & 1 - |w_3|^2 \end{bmatrix}$$

□

$$\begin{aligned} U \text{ is Hermitian: } U^\dagger &= (I - 2ww^\star)^\dagger = I^+ - 2(ww^\star)^\dagger \\ &= I - (2w^\star)^\star w^\star = I - 2ww^\star = U \end{aligned}$$

U is orthogonal: $U^\dagger U = U^2 = (I - 2ww^\star)^2 = I - 4ww^\star + 4(ww^\star)(ww^\star) = I$ because $w^\star w = 1$ and the association law applies, then

$$(ww^\star)(ww^\star) = w(w^\star w)w^\star = ww^\star$$

□

The QR Factorization of a Matrix

$$A = QR.$$

Take $A \in \mathbb{C}^{n\times n}$. Let $P_r = I - 2w^{(r)}w^{(r)\dagger} \quad r = 1, \cdots, n-1$, where $\dagger$ indicates adjoint.

It turns out that we want to enter zeros in the definition of the w's:

(85) $w^{(r)} = [0, 0, \cdots, 0, w_r, \cdots, w_n]^T \equiv [O_{r-1}, \widehat{w}^T]^T$ with $\widehat{w} \in \mathbb{C}^{n-r+1}$

Writing A in terms of its columns $A_{\star 1,} \cdots A_{\star n}$ we have

$$P_1 A = [P_1 A_{\star 1}, \ldots, P_1 A_{\star n}].$$

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 $b \neq \mathbf{0}$ be given, $b \in \mathbb{C}^n$ and suppose we want to produce U of the form such that $U\mathbf{b}$ contains zeros in positions $r+1$ through n, for some $r \geq 1$. Choose w as in (85). Then the first $r-1$ elements of $\mathbf{b}$ and $U\mathbf{b}$ are the same.

Let's write $m \equiv n - r + 1$

(86) $$\text{so we take } w = \begin{bmatrix} O_{r-1} \\ v \end{bmatrix} \text{ and}$$

(87) $$A_{\star 1} = \begin{bmatrix} c \\ d \end{bmatrix}$$

$c \in \mathbb{R}^{r-1}$ and $v, d \in \mathbb{R}^m$. Then our restriction on the form of $U\mathbf{b}$ requires the $r-1$ components of $U\mathbf{b}$ and c be same, and

(88) $$(I - 2vv^T)d = [\alpha, 0, 0 \ldots 0]^\dagger \quad ||v||_2 = 1$$

for some α. Since $I - 2vv^T$ is orthogonal, the length of d is preserved $\therefore$

$$|\alpha| = ||d||_2 \equiv S$$

(89) $$\alpha = \pm S = \pm\sqrt{d_1^2 + d_2^2 \cdots + d_m^2}$$

Let

$$p = v^T d$$

from (88)

$$d - 2pv = [\alpha, 0, \ldots, 0]^T \tag{90}$$

Multiplication by v^T and use of $||v||_2 = 1$ implies

$$p = -\alpha v_1 \tag{91}$$

Substituting this into the first component of (90) 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 α in (89) 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, obtain p from (91). Return to (90) and then using components $2 - m$

$$v_j = \frac{d_j}{2p} \quad j = 2, 3, \cdots, m.$$

Once v is obtained $\Rightarrow$ w and U are obtained. The operation count can be shown to be $O((2m + 2))$. □

Now $P_1 A$ contains zero below the diagonal in its first column. Choose P_2 similarly, so that $P_2 P_1 A$ contains zeros in its second column below diagonal. Note that $P_1 A$ and $P_2 P_1 A$ have same elements in row 1 and column 1. So to get P_2 and $w^{(2)}$ replace $A_{\star 1}$ in (87) by second column of $P_1 A$. Continue procedure till we obtain an upper triangular matrix

$$R \equiv P_{n-1} P_{n-2} \cdots P_1 A.$$

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 $A = QR$. □

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

$$\begin{aligned} A_m &= Q_m R_m \\ A_{m+1} &= R_m Q_m \\ m &= 1, 2, \cdots \end{aligned}$$

Since $R_m = Q_m^T A_m \Rightarrow$

$$A_{m+1} = Q_m^T A_m Q_m$$

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).

□

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 unitary similarity transformations. An "upper Hessenberg Matrix" by unitary similarity transformations. An "upper Hessenberg" matrix has the form:

$$H = \begin{bmatrix} \star & \star & - & - & - & \star & \star \\ \star & & & & & & : \\ 0 & \star & & & & & : \\ & 0 & \star & & & & : \\ : & & 0 & \star & & : & \\ & & & 0 & \star & \star\star & \\ 0 & & & & 0 & \star\star & \end{bmatrix} \text{ i.e. } h_{ij} = 0 \text{ when } 1 > j + 1.$$

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: Let A be real $n \times n$ and eigenvalues $\{\lambda_i\}$

$$|\lambda_1| > |\lambda_2| \cdots |\lambda_n| > 0 \tag{92}$$

then the iterates R_n of 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 (92), 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 2×2 matrices. Thus the eigenvalues of A can be easily computed from those of D. For A non-symmetric, the situation is more complicated but still acceptable.

□

13.5 Inverse Iteration

To find 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 λ be an approximation to a simple eigenvalue λ_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 λ to be a "poor" guess of λ_k, since otherwise we get a severely ill-conditioned matrix $A - \lambda I$! So choose a "close" value.

More precisely: let

$$z^{(0)} = \sum_{i=1}^{n} \alpha_i x_i, \qquad \alpha_k \neq 0. \tag{93}$$

from the power method:

$$z^{(m)} = \frac{\sigma_m (A - \lambda I)^{-m} z^{(0)}}{||(A - \lambda I)^{-m} z^{(0)}||_\infty} \tag{94}$$

(95)

$$|\sigma_m| = 1 \text{ i.e. either } \pm 1. \tag{96}$$

substituting (93):

$$(A - \lambda I)^{-m} z^{(0)} = \sum_{i=1}^{n} \alpha_i [\frac{1}{\lambda_i - \lambda}]^m x_i \tag{97}$$

let $\lambda_1 - \lambda = \varepsilon$ and assume $|\lambda_1 - \lambda| \geq c > 0 \quad i = 1, 2, \ldots, n \quad i \neq k$.

¿From (96) and (97)

$$z^{(m)} = \sigma_m \frac{x_k + \varepsilon^m \sum_{i \neq k} \frac{\alpha_i}{\alpha_k} [\frac{1}{\lambda_i - \lambda}]^m x_i}{||x_k + \varepsilon^m \sum_{i \neq k} \frac{\alpha_i}{\alpha_k} [\frac{1}{\lambda_i - \lambda}]^m x_1||_\infty}. \tag{98}$$

If $|\varepsilon| < c$ then

$$||\varepsilon^m \sum_{i \neq k} \frac{\alpha_i}{\alpha_k} [\frac{1}{\lambda_i - \lambda}]^m x_i||_\infty \leq \left[\frac{\varepsilon}{c}\right]^m \sum_{i \neq k} |\frac{\alpha_i}{\alpha_k}|$$

which tends to 0 as $m \to \infty$, and with (98) 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 $A - \lambda I$.

□

14 DIFFERENCE EQUATIONS

Let V denote a set containing an infinite sequence of complex numbers. For example,

$$\begin{aligned} x &= [x_1, x_2, x_3 \cdots] \text{ and} \\ y &= [y_1, y_2, y_3 \cdots] \quad \text{are 2 elements of } V. \end{aligned}$$

V also contains $\mathbf{0} = [0, 0, 0 \cdots]$.

On V we define the 2 operations:

$$\begin{aligned} x + y &= [x_1 + y_1, x_2 + y_2, y_3 + y_3, \ldots] \\ \lambda x &= [\lambda x_1, \lambda x_2, \ldots] \end{aligned}$$

compactly:

$$\begin{aligned} (x+y)_n &= x_n + y_n \\ (\lambda x)_n &= \lambda x_n. \end{aligned}$$

If it scales, adds, has a $\mathbf{0} \Rightarrow V$ is an infinite-dimensional vector space. It is spanned by

$$\begin{aligned} V^1 &= [1, 0 \cdots] \\ V^2 &= [0, 1, 0 \cdots] \\ V^3 &= [0, 0, 1 \cdots] \\ &\vdots \end{aligned}$$

definition: Linear operator: $L : V \to V$

Example Displacement operator

$$\begin{aligned} \text{let } x &= [x_1, x_2, x_3, \cdots] \\ \text{then } Ex &\equiv [x_2, x_3, x_4, \cdots] \\ (Ex)_n &\equiv x_{n+1} \\ (EEx)_n &= (Ex)_{n+1} = x_{n+2} \\ (E^k x)_n &= x_{n+k} \end{aligned}$$

□

Example Linear Differential operator (with constant coefficients and finite rank)

$$\begin{aligned} L &= \sum_{i=0}^{m} c_i E^i \\ E^0 &= \text{is identity operator } (E^0 x)_n = x_n \end{aligned}$$

L is a polynomial in E; can write as powers in E:

$$\begin{aligned} L &= p(E) \\ p(\lambda) &= \sum_{i=0}^{m} c_i \lambda^i \text{ characteristic polynomial.} \end{aligned}$$

Want to study $Lx = 0$

the set $\{x : Lx = 0\}$ is a linear subspace of V.

It is called the NULL SPACE.

$Lx = 0$ is solved if we know a basis for the null space of L.

take $c_0 = 2 \quad c_1 = -3 \quad c_2 = 1$

$$\begin{aligned} L &= E^2 - 3E^1 + 2E^0 \\ Lx &= (E^2 - 3E^1 + 2E^0)x = 0 \\ x_{n+2} - 3x_{n+1} + 2x_n &= 0 \quad n \geq 1 \\ p(\lambda) &= \sum_{i=0}^{2} c_i \lambda^i = \lambda^2 - 3\lambda + 2 \\ p(E)x &= 0 \end{aligned}$$

Put $x_n = \lambda^n \quad \Rightarrow \quad \lambda^n - 3\lambda^{n+1} + 2\lambda^n = 0$

$$\lambda^n(\lambda^2 - 3\lambda + 2) = 0$$

characteristic equation: $\lambda^n(\lambda - 2)(\lambda - 1) = 0$ has solutions

$[0, 0, 0 \cdots]$, $v_n = 2^n$ and $u_n = 1^n$. The first one is the trivial solution, the other two are SIMPLE zeros.

It turns out that $u = u_n \quad v = v_n$ form a basis: To see this:

$$x = \alpha u + \beta v$$
$$x_n = \alpha u_n + \beta v_n \quad \forall n$$

(99) $$\text{if } n = 1 \text{then} \quad \begin{cases} x_1 = \alpha + 2\beta \\ x_2 = \alpha + 4\beta \end{cases}$$

(99) determines α, β since $\det \begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} \neq 0$ (This is a Vandermode Matrix).

$$\begin{aligned} x_n &= 3x_{n-1} - 2x_{n-2} \\ &= 3(\alpha u_{n-1} + \beta u_{n-1}) - 2(\alpha u_{n-2} - \beta v_{n-2}) \\ &= \alpha(3u_{n-1} - 2u_{n-2}) + \beta(3v_{n-1} - 2v_{n-2}) \\ &= \alpha u_n + \beta v_n \end{aligned}$$

THIS WORKS WHEN CHARACTERISTIC EQUATION HAS SIMPLE ZEROS.

Theorem: If p is the characteristic polynomial and λ is a zero of $p \Rightarrow$ one solution of $p(E)x = 0$ is $[\lambda, \lambda^2, \lambda^3 \cdots]$. If all zeros of p are simple and nonzero $\Rightarrow$ each solution of the difference equation is a linear combination of such special solutions.

□

What to do when $p(E)x = 0$ when p has non-simple zeros?

Zeros of Higher Multiplicity:

$$\begin{aligned} x(\lambda) &= [\lambda, \lambda^2, \lambda^3 \cdots] \\ p(E)x(\lambda) &= p(\lambda)x(\lambda) \end{aligned}$$

differentiate with respect to λ

$$p(E)x'(\lambda) = p'(\lambda)x(\lambda) + p(\lambda)x'(\lambda)$$

if λ is a multiple zero of p, then $p(\lambda) = p'(\lambda) = 0 \therefore x(\lambda)$ and $x'(\lambda)$ solve 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 sequence truncated to 2^{nd} term the resulting vectors in $\mathbb{R}^2$ are linearly independent.

Same reasoning shows that if λ is a zero of p having k multiplicity

$$\begin{aligned} p(E)x &= 0 \text{ has solutions} \\ x(\lambda) &= [\lambda, \lambda^2, \lambda^3 \cdots] \\ x'(\lambda) &= [1, 2\lambda, 3\lambda^2 \cdots] \\ x''(\lambda) &= [0, 2, 6\lambda \cdots] \\ \vdots &= \vdots \\ x^{k-1}(\lambda) &= \frac{d^{k-1}}{d\lambda^{k-1}}[\lambda, \lambda^2, \lambda^3 \cdots] \end{aligned}$$

Theorem II: 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 $x(\lambda), x'(\lambda), x''(\lambda) \cdots x^{k-1}(\lambda)$

$$\text{where } (\lambda) = [\lambda, \lambda^2, \lambda^3 \cdots]$$

i.e. general solution if λ_0 is one such non-simple zeros, the the general solution is $\alpha_1 x(\lambda_0) + \alpha_2 x'(\lambda_0) + \ldots \alpha_{k-1} x^{k-1}(\lambda_0)$ $\square$

Example $4x_n - 9x_{n-1} + 6x_{n-2} - x_{n-3} = 0$

$$\begin{aligned} &\text{let } x_n = \lambda^n \Rightarrow \lambda^n(4 - 9\lambda^{-1} + 6\lambda^{-2} - \lambda^{-3}) = 0 \\ &\lambda^n(4\lambda^3 - 9\lambda^2 + 6\lambda - 1) = 0 \\ &\lambda^n(\lambda + 1)^2(4\lambda - 1) = 0 \end{aligned}$$

The zeros of the characteristic are: -1 (double, $k = 2$), $1/4$ (simple).

$$\begin{aligned} x(\lambda) &= [\lambda, \lambda^2, \lambda^3 \cdots] \\ x'(\lambda) &= [1, 2\lambda, 3\lambda^2 \cdots] \end{aligned}$$

$$x(-1) = [-1, 1, -1 \cdots], \quad x\left(\frac{1}{4}\right) = \left[\frac{1}{4}, \frac{1}{4^2}, \frac{1}{4^3}, \cdots\right]$$
$$x'(-1) = [1, -2, 3, -4 \cdots]$$

general solution: $x = \alpha x(-1) + \beta x'(-1) + \lambda x\left(\frac{1}{4}\right)$

or $x_n = \alpha(-1)^n + \beta n(-1)^{n-1} + \lambda\left(\frac{1}{4}\right)^n$ □

Example Consider $x_{n+1} - nx_n = 0$, not polynomial.

By inspection:

$n = 1 \quad x_2 = x_1$

$n = 2 \quad x_3 = 2x_2 = 2x_1$

$n = 3 \quad x_4 = 3x_3 = 3 \cdot 2x_1$
$x_n = (n-1)!x_1$

□

Stable Difference Equations

$x = [x_1, x_2 \cdots x_n \cdots] \in V$

is bounded if $\exists\, c$ constant such that $|x_n| \leq c \,\forall\, n$

i.e. $\sup_n |x_n| < \infty$

$p(E)x = 0$ is stable if all its solutions are bounded.

Theorem for p with $p(0) \neq 0$

equivalent $\begin{cases} \text{(i) } p(E)x = 0 \text{ is stable} \\ \text{(ii) All zeros of } p \text{ satisfy } |z| \leq 1 \text{ for simple zeros} |z| < 1 \text{ for zeros of higher multiplicit} \end{cases}$

Example $4x_n + 7\lambda x_{n-1} + 2x_{n-2} - x_{n-3} = 0$
$4\lambda^3 + 7\lambda^2 + 2\lambda - 1 \to p$ has -1 double and $\frac{1}{4}$ $\therefore$ unstable.

Example $x_n = x_{n-1} + x_{n-2} \quad \lambda^2 - \lambda 4 = 0$ roots:

$\left(1 \pm \sqrt{5}\right)/2$ $\therefore$ unstable.

□