Notes on Numerical Analysis

David Karapetyan

Contents

1 Limitations of the Machine

A computer is just a sophisticated collection of circuits with "off" and "on" switches. We represent "off" and "on" by 0 and 1, respectively. A computer program at its lowest level is just a collection of 0's and 1's. When we write a computer program in a static programming language such as C, a compiler translates the program into 0's and 1's, which are then then processed by the computer CPU.

Besides the base 10 system, it is important to understand binary (base 2), octal (base 8), and hexadecimal (base 16). Binary is encountered most often, since that is the number system computer instructions ("on" and "off" switches) are encoded in.

1.1 Loss of Precision

Each digit in a binary representation of a number takes a *bit* of memory to hold. (There are 8 bits in a byte, 1000 bytes in a megabyte, 1000 megabytes in a gigabyte, and so on). Since there is a finite amount of memory in the computer, we must chop all digits after a specific digit in a binary number.

Example. $1/10 = (0.001100110010011\ldots)_2 \approx (0.00110011)_2$

If we chop at the nth digit, then the error bound is 10^{-n}. We have the alternative of rounding up or down after the nth digit. In this case the error bound is $1/2 \times 10^{-n}$.

For binary, we have the scientific notation $x = q \times 2^m$, where $1/2 < |q| < 1$. This is entirely analogous to the base 10 case.

In these days of increasing RAM and decreasing prices, one need not be too worried about truncation or rounding errors, unless the numerical simulation one is performing is highly unstable (in which case, small errors can lead to wildly incorrect results—more on this later).

1.2 Loss of Significance

Definition. Let $x^\star \in \mathbb{C}$ be an approximation to $x \in \mathbb{C}$. Then we define the *absolute error* and the *relative error* as $|x - x^\star|$ and $|(x - x^\star)/x|$, respectively.

In practice, absolute error matters very little, and can be incredibly misleading.

Example. Michael donates $\$10,000$ to a charity. He is hailed as benevolent. His annual salary is $\$10,000,000$. In absolute terms, this seems like a lot. In relative terms, he has donated $\$10,000/10,000,000 \times 100\%$, or 0.1% of one year's salary. For perspective, if Joe's annual salary is $\$60,000$, then 0.1% of this is $\$60$.

Example. Take $x = 10^6 + 10^3$ and $x^\star = 10^6$. Then $x - x^\star = 10^3$, but $(x - x^\star)/x = 10^{-3}$.

In short, relative errors are much more important. Loss of significance occurs when two nearly equal quantities are subtracted.

Example. Let $x = 0.3721478613$ and $y = 0.3720230572$. Then $x - y = 0.0001248121$. The repeating $0's$ are taking up memory that could be better used storing significant digits (digits other than 0). If our computer truncates after the $6th$ digit, then $x^\star = 0.37215$, $y^\star = 0.37202$, and the relative error of $x^\star - y^\star$ from $x - y$ is approximately 4%. This is because $x^\star - y^\star = 0.00013$. We have wasted three digits out of five digits on storing $0's$.

Hence, if possible, one should avoid subtracting nearly equal quantities in an algorithm.

Example. If $y = \sqrt{x^2+1} - 1$ and we are inputting $x \approx 1$, we rewrite this as $x^2/\sqrt{x^2+1} + 1$. No loss of significance occurs in this latter form.

A loss of precision also occurs when dividing by numbers very close to 0. Indeed, we now demonstrate the link between subtracting two nearby numbers and division by a number close to 0.

Example. Let x and $y \approx 0$ be real, strictly positive numbers with a decimal component, and let $x^\star, y^\star$ be their truncated machine approximations, where the truncation occurs after the $5th$ decimal digit. We compute the relative error of $x^\star/y^\star$ from x; that is

$$\left| \frac{x/y - x^\star/y^\star}{x/y} \right| = \left| \frac{x - x^\star y/y^\star}{x} \right|$$

Observe that, for as long as $y/y^\star$ remains defined, it approaches $\pm\infty$ as $y \to 0$. Therefore, our relative error, while defined, approaches ∞.

As a last example, observe that even if we are careful not to subtract nearby quantities or divide by numbers close to 0, we can still have precision errors if we are inputting approximated values into highly oscillatory functions.

Example. Consider the function $f(x) = \cos(nx)$, where $n \gg 1$ is an integer. Then $f(2\pi) = 1$, but $f(2\pi^\star)$ can be quite far from 1. One way around this is to use interval arithmetic—that is, try to compute what interval values near 2π will get mapped to. This is outside the scope of this course.

One way to partially offset all these worries about loss of precision and significance is to have a lot of RAM, and to take advantage of 64 bit processors (which will eventually become 128 bit, 256 bit, and so on.

1.3 Numerical Instability

There are certain numerical iterations that are more sensitive to loss of precision and significance than others.

Example. Consider

$$\begin{aligned} x_{n+1} &= 13x_n/3 - 4x_{n-1}/3 \\ x_0 &= 1 \\ x_1 &= 1/3 \end{aligned}$$

This has the theoretical solution $x_n = (1/3)^n$. However, x_n diverges to infinity numerically. This is because of the loss of precision emanating from repeated subtractions of nearby quantities. These errors accumulate with each iteration. We call such a numerical iteration *unstable*.

Remark. A stable version of the same iteration is given by

$$\begin{aligned} &2x_n/3 - x_{n-1}/3 \\ &x_0 = 1 \\ &x_1 = 1/3 \end{aligned}$$

We can extend our notion of instability to functions as well. For example, we can ask how sensitive a function's output value is to perturbations of its input. If f is differentiable, we compute the relative error of f at x:

$$\frac{f(x+h) - f(x)}{f(x)} \approx \frac{hf'(x)}{f(x)} = \left[\frac{xf'(x)}{f(x)}\right]\left(\frac{h}{x}\right)$$

which motivates the following definition.

Definition. Let I be an open subset of $\mathbb{R}$. Then for $f \in C^1(I)$, we call

$$\left[\frac{xf'(x)}{f(x)}\right]$$

the *condition number* of f in I.

Note that the condition number can potentially magnify the relative error h/x of the perturbed input, resulting in a potentially large relative error for the output. Heuristically, the condition number is a measure of how sensitive f is to perturbations in its input.

Example. Let $f, g \in C^2$, $F = f + \varepsilon g$, and r a simple root of $f \in C^2$. Then we must perturb $r \mapsto r + h$ to obtain the corresponding root of F. We are interested in the magnitude of h. We have

$$\begin{aligned} 0 = F(r+h) &\approx f(r) + hf'(r) + \varepsilon g(r) + h\varepsilon g'(r) + O(h^2) \\ &= h[f'(r) + \varepsilon g'(r)] + \varepsilon g(r) + O(h^2) \end{aligned}$$

Assuming $h, \varepsilon \ll 1$, we obtain

$$h \approx -\frac{\varepsilon g(r)}{f'(r)}.$$

Letting $f(x) = \prod_{k=1}^{20}(x-k)$ and $g(x) = x^{20}$, we obtain

$$h \approx -\varepsilon g(20)/f'(20) = -\varepsilon 20^2 0/20! \approx -\varepsilon 10^8.$$

This suggests that even for small ε, if $\varepsilon \not< 10^{-8}$, then h may potentially be very large.

2 Orders of Convergence

We'd like to understand how fast a sequence converges to its limit, without having to worry about silly constants.

Definition. Let $\{x_n\} \in \mathbb{C}$ be a sequence converging to x. We say that $x_n = O(\alpha_n)$ if there exists integer $N \geq 0$ such that $|x_n| \leq C|\alpha_n|$ for $n \geq N$, and $x_n = o(\alpha_n)$ if $|x_n| \leq \varepsilon_n|\alpha_n|$, where $\varepsilon_n \to 0$. Similarly, for functions $f, g : \mathbb{C} \to \mathbb{C}$, we say that $f = O(g)$ in a neighborhood N of x if $|f(x_n)| \leq C|g(x_n)|$ for all $x_n \in N$, and $f = o(g)$ in a neighborhood N of x if $|f(x_n)| \leq \varepsilon_n|g(x_n)|$, where $\varepsilon_n \to 0$, for all $x_n \in N$.

Example.

$$\frac{n+1}{n^2} \leq \frac{2n}{n^2} = \frac{2}{n} = O\left(\frac{1}{n}\right)$$

Example.

$$\frac{1}{n \log n} \leq \frac{\varepsilon_n}{n} = o\left(\frac{1}{n}\right).$$

Example.

$$\frac{5}{n} + e^{-n} = O\left(\frac{1}{n}\right)$$

Remark. It is a true statement that $5/n^4 = O(1/n)$. However, this grossly misrepresents the asymptotic behavior of the sequence $5/n^4$. Typically, when we discuss the order of convergence of a sequence x_n to its limit, we seek the sharpest order α_n of convergence. By sharp, we mean that if $|x_n| = O(\alpha_n)$, then $|x_n| \neq O(\alpha_n n^{-\varepsilon})$, for any $\varepsilon > 0$.

Example. Using Taylor series,

$$e^x - \sum_{k=0}^{n-1} \frac{1}{k!} x^k = \frac{1}{n!} f^{n+1}(\xi) x^n = O\left(\frac{1}{n!}\right).$$

3 Linear Difference Equations and Operators

Algorithms emit sequences. If we can detect a pattern between successive iterates that is linear in nature, we can use powerful tools from linear algebra to study the computational complexity and convergence properties of our algorithm.

Definition. Denote V to be the set of all infinite complex sequences

$$\vec{x} = [x_1, x_2, \ldots, x_n, \ldots]$$

equipped with addition

$$\vec{x} + \vec{y} = [x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n, \ldots],$$

scalar multiplication

$$\lambda \vec{x} = [\lambda x_1, \lambda x_2, \ldots, \lambda x_n, \ldots],$$

and the norm

$$\|\vec{x}\| = \sup\{|x_i|\}$$

It is not hard to see that V is an separable infinite-dimensional vector space.

We wish to study the vector space $\mathcal{L}(V)$ of bounded linear operators $T : V \to V$, equipped with the standard definitions for addition, scalar multiplication, and operator norm. In particular, we are interested in the shift operator

$$E : V \to V,$$
$$E\vec{x} = [x_2, x_3, \ldots, x_{n+1}, \ldots]$$

which has the property

$$(E^k x)_n = x_{n+k}.$$

Definition. Observe that the set of shift operators E_k form a basis for a linear subspace of $\mathcal{L}(V)$. We call this the space of *linear difference operators*, and denote it by $\mathcal{D}(V)$.

Definition. Consider an element $T = \sum_{i=0}^{m} c_i E^i \doteq p(E)$ of $D(V)$. We call $p(x) = \sum_{i=0}^{m} c_i x^i$ the *characteristic polynomial* of T.

Remark. Studying a linear difference equation is equivalent to studying the null space of some operator $T \in D(V)$. Since V is separable, we can think of $\mathcal{D}(V)$ heuristically as nothing more than the space of $\mathbb{N} \times \mathbb{N}$ matrices.

Example.

$$x_{n+2} - 3x_{n+1} + 2x_n = 0 \longleftrightarrow (E^2 - 3E^1 + 2)\vec{x} = 0 \longleftrightarrow p(E)\vec{x} = 0, \quad p(\lambda) = \lambda^2 - 3\lambda + 2.$$

The tools from finite-dimensional matrix algebra extend to the countable-dimension case. More precisely, we can categorize the solutions of the of certain finite difference problem entirely in terms of eigenvectors and eigenvalues.

Theorem 1. *Let $p(E)\vec{x} = 0$ be a linear difference equation.*

1. *If λ is a root to its characteristic polynomial, then $\vec{u} = [\lambda, \lambda^2, \lambda^3, \ldots]$ is a solution to $p(E)\vec{x} = 0$.*
2. *If all roots λ_k of p have multiplicity 1 and are nonzero, then $\bigcup_k \{[\lambda_k, \lambda_k^2, \ldots]\}$ is a basis for the solution space of $p(E)\vec{x} = 0$.*
3. *If all roots λ_k of p are nonzero, and have multiplicity m_k greater than or equal to 1, then $\bigcup_k \left\{[\lambda_k, \lambda_k^2, \ldots], [\lambda_k, \lambda_k^2, \ldots]', \ldots, [\lambda_k, \lambda_k^2, \ldots]^{(m_k - 1)}\right\}$ is a basis for the solution space of $p(E)\vec{x} = 0$.*

Proof. We split the proof into two parts.

1. We first show that $\vec{u}$ is an eigenvector of $p(E)$ with corresponding eigenvalue $p(\lambda)$. Observe that

$$\begin{aligned} p(E)\vec{u} = \left(\sum_{i=0}^{m} c_i E^i\right)\vec{u} &= \sum_{i=0}^{m} c_i \left(E^i \vec{u}\right) \\ &= \sum_{i=0}^{m} c_i [\lambda^{i+1}, \lambda^{i+2}, \ldots] \\ &= \sum_{i=0}^{m} c_i \lambda^i \vec{u} \\ &= p(\lambda)\vec{u}. \end{aligned}$$

   Hence, if $p(\lambda) = 0$, $\vec{u}$ is a solution.

2. For each root λ_k, consider the corresponding solution $u^{(k)} = [\lambda_k, \lambda_k^2, \ldots]$. Let $\vec{x}$ be an arbitrary solution to our linear finite difference problem. We want to show that $\vec{x} = \sum_{k=1}^{m} a_k u^{(k)}$. We can write the first m components of $\vec{x} = [x_1, x_2, \ldots]$ as

$$x_i = \sum_{k=1}^{m} a_k \lambda_k^i, \quad 1 \leq i \leq m. \tag{3.1}$$

   Writing this in matrix form, we obtain

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \underbrace{\begin{bmatrix} \lambda_1 & \lambda_2 & \ldots & \lambda_m \\ \lambda_1^2 & \lambda_2^2 & \ldots & \lambda_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^m & \lambda_2^m & \ldots & \lambda_m^m \end{bmatrix}}_{\text{A}} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

   Observe that if we factor λ_1^j from row j, for each $1 \leq j \leq m$, we obtain a Vandermonde matrix, which is invertible, by Lemma 7. Hence, A is invertible, and so completely determines the a_k, for $1 \leq k \leq m$. As a corollary of this argument, we obtain that the vectors $u^{(k)}$, $1 \leq k \leq m$ are linearly independent. A simple induction on m completes the proof.

3. In the case where roots of multiplicity greater than 1, we have more difficulties: if we construct a matrix A as in the proof of Theorem 1, it will be singular (we will have columns that are identical). Hence, we must somehow construct new solutions out of roots with multiplicity greater than 1. We do so as follows. Assume λ has multiplicity n. Then for $k \leq n-1$

$$\begin{aligned} p(E)x^{(k)}(\lambda) &= [p(E)x(\lambda)]^{(k)} \\ &= [p(\lambda)x(\lambda)]^{(k)} \\ &= \sum_{i=0}^{k} \binom{k}{i} p^{(k-i)}(\lambda) x^{(i)}(\lambda) = 0. \end{aligned}$$

Assume the set $\left\{x(\lambda)^{(k)}\right\}_{k=0}^{n-1}$ is not linearly independent. Then

$$x^{(n-1)}(\lambda) = \sum_{i=0}^{n-2} c_i x^{(i)}(\lambda)$$

where some $c_i \neq 0$. By differentiation

$$x^{(n)}(\lambda) = \sum_{i=0}^{n-1} c_i x^{(i)}(\lambda)$$

and hence

$$p(E)x^{(n)}(\lambda) = 0$$

which implies $x(\lambda)$ has multiplicity greater than n, which is a contradiction. Lastly, we run an argument similar to that in (2) to complete the proof, with the observation that we can now fill the "redundant" columns generated by non-simple roots with the new vectors we construct out of their derivatives.

□

Lastly, we would like to identify whether or not solutions to a finite difference problem have the potential to blow up.

Definition. We say $x \in [x_1, x_2, \ldots, x_n, \ldots]$ is *bounded* if $\sup_n |x_n| < \infty$. A difference equation $p(E)x = 0$ is *stable* if all its solutions are bounded.

Theorem 2. *For a polynomial p with $p(0) = 0$, $p(E)x = 0$ is stable if and only if all simple roots of p satisfy $|\lambda| \leq 1$, and all non-simple roots satisfy $|\lambda| < 1$.*

Proof. To prove necessity, we observe that if $|\lambda| > 1$ for some simple root, or $|\lambda| \geq 1$ for some non-simple root, then $[\lambda, \lambda^2, \lambda^3, \ldots]$ or $[\lambda, \lambda^2, \lambda^3, \ldots]'$ are unbounded, respectively. To prove sufficiency, we observe that

$$[\lambda, \lambda^2, \lambda^3, \ldots]^{(n)} = \left[\frac{n!}{(n-k)!}\lambda^{n-k}\right]_{n=k}^{\infty}$$

is bounded for all $n \geq 0$ if $|\lambda| < 1$. This follows immediately from the estimate

$$\frac{n!}{(n-k)!}\lambda^{n-k} \leq n^k \lambda^{n-k} \to 0 \text{ as } n \to \infty$$

where the right hand side decays to 0 as $n \to \infty$ if $|\lambda| < 1$. Lastly, every solution to $p(E)x = 0$ is a finite linear combination of vectors of form $[\lambda, \lambda^2, \lambda^3, \ldots]^{(n)}$ by Theorem 1. □

Example. The difference equation $4x_n + 7x_{n-1} + 2x_{n-2} - x_{n-3} = 0$ is unstable, because its characteristic polynomial $4\lambda^3 + 7\lambda^2 + 2\lambda - 1$ has a root $\lambda = -1$ of multiplicity 2.

4 Finding Roots of Functions

We are interested in finding x such that $f(x) = a$, where $a \in \mathbb{R}$. This is equivalent to finding the roots of the function $g(x) \doteq f(x) - a$.

4.1 Bisection Method

We assume that f is continuous. Then if $f(a)f(b) \leq 0$ for $a < b$, then we must have $f(x) = 0$ for some $x \in [a, b]$. Assume this is true. We then split $[a, b]$ into the intervals $[c_0, c_1]$ and $[c_1, c_2]$, where $c_0 = a$, $c_2 = b$, and $c_1 = (a+b)/2$. Then either $f(c_0)f(c_1) \leq 0$ or $f(c_1)f(c_2) \leq 0$. Assume without loss of generality that $f(c_0)f(c_1) \leq 0$. Then we split the interval $[c_0, c_1]$ at its midpoint, and repeat the procedure. We now show that this procedure eventually yields a root of f.

Convergence and Error Analysis

We would like to first establish that our method converges to a root of $f(x)$ in the interval $[a, b]$, given that f has a root in this interval. Now $\{a_n\}$ and $\{b_n\}$ are increasing, and decreasing sequences, respectively, both of which are bounded from below and above by a and b. Hence, both converge to a limit. Indeed,

$$b_{n+1} - a_{n+1} = 1/2(b_n - a_n) \tag{4.1}$$

and so

$$b_n - a_n = 2^{-n}(b_0 - a_0) \tag{4.2}$$

which implies

$$\lim a_n = \lim b_n = c.$$

However, a_n and b_n have the property that

$$f(a_n)f(b_n) \leq 0.$$

By continuity

$$0 \geq \lim f(a_n)f(b_n) = f(c)^2$$

which implies that $f(c) = 0$. Hence, the bisection method has converged to a root c in $[a, b]$. Hence, one approach to approximating c is to prescribe an error threshold, and take a_n, for n sufficiently large. However, we can get a better estimate if we take $c_n \doteq 1/2(a_n + b_n)$. By triangle inequality and (4.1), we obtain

$$|c - c_n| \leq (b_n - a_n)/2$$

and so

$$|c - c_n| \leq 2^{-n-1}(b_0 - a_0).$$

Compare this with the estimate (4.2). It guarantees convergence that is at least twice as fast as that given by taking a_n or b_n as our approximation to c. Note that the speed of convergence is linear.

4.2 Newton's Method

The main advantages of Newton's method over the bisection method is that if it converges to a root, then its convergence is quadratic. The drawbacks is that it may not converge to a root at all if the starting point x_0 of the algorithm is not sufficiently close to a root. (In particular, a general heuristic is that if the function becomes steep near a root, then our choice x_0 has to be very close to that root in order to converge to it.) The essence of the Newton algorithm is to approximate a function via its linearization. More precisely, assuming r is a root of $f \in C^2$, and x_0 is an initial guess a positive distance h from r, we write

$$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0) + O(h^2)$$

from which we obtain $h \approx -f(x_0)/f'(x_0)$. Hence

$$\begin{aligned} r &\approx x_0 + h \\ &\approx x_0 - \frac{f(x_0)}{f'(x_0)}. \end{aligned}$$

The Newton algorithm is then given by

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

Error Analysis

Assume we have a suitable $x_0 \in I$ and $f \in C^2(I)$ such that the algorithm converges to a *simple* root $r \in I$. Then we have

$$e_{n+1} = x_{n+1-r} = e_n - f(x_n)/f'(x_n) = \frac{e_n f'(x_n) - f(x_n)}{f'(x_n)}$$

and

$$0 = f(r) = f(x_n - e_n) = f(x_n) - e_n f'(x_n) + e_n^2 f''(\xi_n)/2.$$

Substituting, we obtain

$$|e_{n+1}| = \left| \frac{e_n^2 f''(\xi_n)}{2f'(x_n)} \right|$$

Since r is a simple root by assumption, we can always choose a subsequence $\{x_{n_k}\}$ such that $f'(x_{n_k}) > c > 0$. Hence, using this fact that the continuity of f'', we obtain the estimate

$$|e_{n+1}| \lesssim |e_n|^2.$$

Hence, if the Newton method converges to a simple root for a C^2 function, it converges quadratically.

4.3 Secant Method

Note that for each iteration of Newton's method, we have two costly computations: computing $f(x_0)$ and $f'(x_0)$. To reduce this cost, we introduce the approximation

$$f'(x_0) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

and substitute this into Newton's method to get

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

Observe that now we have only one costly computation per iteration. That is, we only need to compute $f(x_n)$, since $f(x_{n-1})$ is known from the previous iteration.

Error Analysis

We write

$$\begin{aligned} e_{n+1} = x_{n+1} - r &= \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} - r \\ &= \frac{f(x_n)e_{n-1} - f(x_{n-1})e_n}{f(x_n) - f(x_{n-1})} \\ &= \frac{[f(x_n)/e_n - f(x_{n-1})/e_{n-1}](x_n - x_{n-1})}{[f(x_n) - f(x_{n-1})]/[x_n - x_{n-1}]} e_n e_{n-1} \end{aligned}$$

which by the mean value theorem is bounded by

$$\begin{aligned} &[f(x_n)/e_n - f(x_{n-1})/e_{n-1}][e_n + r - (e_{n-1} + r)]e_n e_{n-1} \\ &= [f(x_n) + f(x_{n-1}) - f(x_{n-1})e_n/e_{n-1} - f(x_n)e_{n-1}/e_n]e_n e_{n-1} \\ &\lesssim e_n e_{n-1} \end{aligned}$$

Therefore,

$$|e_{n+1}| \lesssim |e_n e_{n-1}|.$$

We now make a guess as to the solution of this recurrence. Let $|e_{n+1}| \sim |e_n|^\alpha$, where α is a strictly positive parameter to be determined. Then substituting into the above relation, we must have

$$|e_n|^\alpha \lesssim |e_n| \times |e_{n-1}|$$

which simplifies to

$$|e_n|^{\alpha-1} \lesssim |e_{n-1}|$$

However, our guess $|e_{n+1}| \sim |e_n|^\alpha$ implies $|e_{n-1}| \sim |e_n|^{1/\alpha}$. Substituting this above, we obtain

$$|e_n|^{\alpha-1} \lesssim |e_n|^{1/\alpha}$$

Hence, we must have $\alpha - 1 = 1/\alpha$, or $\alpha^2 - \alpha - 1 = 0$, which gives $\alpha = (1 + \sqrt{5})/2 \approx 1.62$. Hence, the secant method, on its surface, converges slower to a root than Newton's method. However, observe that the computational cost of two iterations of the secant method is equivalent to cost of one iteration of Newton's method (we have one function evaluation at each iteration for the secant method, versus two for Newton's method). Hence, if function evaluations are expensive, and we restrict the number of computations to at most n function evaluations, for some n, then the secant method converges $(2 \times 1.62)/2 = 1.62$ times as fast as Newton's method.

In short, if function evaluations are cheap, use Newton's method. If they are expensive, use the secant method.

4.4 Fixed Points

Observe that Newton's method can be rewritten as

$$x_{n+1} = F(x_n), \quad F(x) = x - f(x)/f'(x).$$

More generally, observe that if arbitrary F is continuous, and $x_n \to s$, then

$$s = \lim x_{n+1} = \lim F(x_n) = F(s)$$

Definition. Let $F : X \to X$, where X is a vector space. We say $s \in X$ is a *fixed point* of F if $F(s) = s$.

Hence, the problem of finding the roots of a function f is equivalent to finding the fixed points of $F(x) \doteq x - f(x)/f'(x)$. What conditions do we need to impose on F in order to guarantee the existence of a fixed point?

Definition. Let (X, d) be a metric space. A mapping $T : X \to X$ is called a *contraction* if there exists an α, $0 \leq \alpha < 1$ such that

$$d(Tx, Ty) \leq \alpha d(x, y), \qquad \forall x, y \in X.$$

Remark. Observe that a contraction mapping is always continuous.

Example. The function $T(x) = 0.1x^2$ defines a contraction mapping in the set $X = [-4, 4]$ equipped with the metric $d(x, y) = |x - y|$.

We now give some examples of iterations with a contractive $F : \mathbb{R} \to \mathbb{R}$.

Example.

$$x_0 = -15$$
$$x_{n+1} = 3 - \frac{x_n}{2}$$

Example.

$$x_0 = \pi/4$$
$$x_{n+1} = 2 - \sin(2x)/3$$

Proposition 3. *Let (X, d) be a complete metric space, and $T : X \to X$ a contraction mapping. Then T has a unique fixed point in X. That is, there is a unique point $x^* \in X$ such that $Tx^* = x^*$. Furthermore, if x_0 is any point in X, and we define the sequence $x_{n+1} = Tx_n$, then $x_n \xrightarrow{X} x^*$ as $n \to \infty$.*

Proof. First we show uniqueness. If x^* and x^{**} are two fixed points, then

$$d(x^*, x^{**}) = d(Tx^*, Tx^{**}) \leq \alpha d(x^*, x^{**}) \implies d(x^*, x^{**}) = 0 \implies x^* = x^{**}.$$

To prove existence, we observe that since X is complete it suffices to show that x_n is Cauchy. A repeated application of the contraction inequality gives

$$\begin{aligned} d\left(x_{n+1}, x_n\right) &= d\left(Tx_n, Tx_{n-1}\right) \\ &\leq \alpha d\left(x_n, x_{n-1}\right) \\ &\leq \alpha^2 d\left(x_{n-1}, x_{n-2}\right) \\ &\cdots \\ &\leq \alpha^n d\left(x_1, x_0\right). \end{aligned}$$

Hence

$$\begin{aligned} d\left(x_{n+k}, x_n\right) &\leq (\alpha^n + \alpha^{n+1} + \cdots + \alpha^{n+k-2} + \alpha^{n+k-1}) d(x_1, x_0) \\ &= \alpha^n (1 + \alpha + \cdots + \alpha^{k-2} + \alpha^{k-1}) \\ &\leq \alpha^n \left(\frac{1}{1-\alpha}\right) \\ &\to 0 \text{ as } n \to \infty \end{aligned}$$

since $0 \leq \alpha < 1$. □

Remark. Note that it is not necessary that F be contractive in order to have convergence of an iteration. For example,

$$\begin{aligned} x_{n+1} &= 1/2 - x_n \\ x_0 &= 1/4 \end{aligned}$$

converges to $1/4$, but $F(x) \doteq 1/2 - x_n$ is not a contraction mapping on any closed subset $I \in \mathbb{R}$. The key thing to realize is that for *special* x_0 we can still have convergence for non-contractive F. The advantage of having contractive F on a closed set X is that we do not need to worry about our initial data—as long as it lives in X, we are guaranteed convergence of our iteration.

It turns out that F also gives great insight into the speed of convergence.

Proposition 4. *Consider the iteration $x_{n+1} = F(x_n)$, where $F \in C^q(I)$ for some integer $q \geq 1$ and open $I \in \mathbb{R}$. Assume F is contractive on a closed subset $I' \subset I$. Then the iteration converges to $s = s(x_0)$, and the order of convergence is the first strictly positive integer $k \leq q$ such that $F^{(k)}(s) \neq 0$.*

Proof. Observe that

$$e_{n+1} = F(x_n) - s = F(s + e_n) - F(s).$$

A Taylor expansion then gives

$$e_{n+1} = e_n F'(s) + e_n^2 F''(s)/2 + \ldots + e_n^{q-1} F^{(q-1)}(s)/(q-1)! + e_n^q F^{(q)}(\xi_n)/q!.$$

Furthermore,

$$e_{n+1} = F(x_n) - F(s) = F'(\xi_n) e_n, \quad s < \xi_n < x_n$$

Since F is contractive on I', it follows from the mean value theorem that

$$|F'(x)| < 1, \quad \forall x \in I'.$$

which guarantees that, for suitably large N, we have $e_n < 1$ for all $n > N$. Assume $k \leq q$ is the first strictly positive integer such that $F^{(k)}(s) \neq 0$. Then from our Taylor expansion, the continuity of $F^{(j)}$, $j \leq q$, and the fact that $e_i < e_i^j$ for any $j \geq 1$, we obtain

$$|e_{n+1}| = O(|e_n|^k)$$

completing the proof. □

5 Roots of Polynomials

Polynomials have much nicer properties than arbitrary functions, so we'd like to use that to our advantage when looking for roots.

Theorem 5. *All zeroes of a polynomial $p(z) = a_0 + a_1 z + \ldots + a_{n-1} z^{n-1} + a_n z^n$, $a_n \neq 0$ lie in a closed disc centered at the origin, with radius*

$$r = 1 + |a_n|^{-1} \max_{0 \leq k \leq n-1} |a_k|.$$

Proof. It is enough to show that there are no zeroes outside this disc. Assuming z is outside the disc, we estimate

$$\begin{aligned}
|p(z)| &\geq |a_n z^n| - |a_{n-1} z^{n-1} + \ldots + a_0| \\
&\geq |a_n||z|^n| - c \sum_{k=0}^{n-1} |z|^k, \quad c \doteq \max_{0 \leq k \leq n-1} |a_k| \\
&= |a_n||z|^n - c \left[\frac{|z|^n - 1}{|z| - 1} \right] \\
&> |a_n||z|^n - c \left[\frac{|z|^n - 1}{1 + |a_n|^{-1} c - 1} \right] \\
&= |a_n||z|^n - |a_n|(|z|^n - 1) \\
&= |a_n| > 0.
\end{aligned}$$

□

Corollary 6. *If all zeroes of a polynomial p are in the disc $\{z : |z| \leq r\}$, then all non-zero zeroes of p are outside the disc $\{z : |z| < 1/r\}$.*

Proof. Let $s(z) \doteq z^n p(1/z)$. Then for nonzero z_0, $s(z_0) = 0$ if and only if $p(1/z_0) = 0$. But for nonzero z_0, $|z_0| \leq r$ is equivalent to $|1/z_0 \geq 1/r|$, which completes the proof. □

5.1 Polynomial Evaluation

For a polynomial $p(z)$, computing the value of $p(z_0)$ by computing $a_2 z^2, a_3 z^3, \ldots, a_n z^n$ in succession is inefficient. The standard method for evaluating a polynomial is via *Horner's algorithm*, which we now derive.

Observe that if p is nth order, then $q(z) = [p(z) - p(z_0)]/(z - z_0)$ is $(n-1)$th order. Let

$$p(x) = a_0 + a_1 z + \ldots + a_n z^n, \quad q(x) = b_0 + b_1 z + \ldots + b_{n-1} b^{n-1}.$$

Then writing $p(z) = (z - z_0)q(z) + p(z_0)$ and substituting, we obtain

$$a_0 + \ldots + a_n z^n = (z - z_0)(b_0 + \ldots + b_{n-1} z^{n-1}) + p(z_0),$$

which admits the following relations

$$\begin{aligned}
a_0 &= p(z_0) - z_0 b_0 \\
a_1 &= b_0 - z_0 b_1 \\
a_2 &= b_1 - z_0 b_2 \\
&\vdots \\
a_{n-1} &= b_{n-2} - z_0 b_{n-1} \\
a_n &= b_{n-1}.
\end{aligned}$$

Rewriting this, we obtain

$$\begin{aligned}
p(z_0) &= a_0 + z_0 b_0 \\
b_0 &= a_1 + z_0 a_1 \\
b_2 &= a_2 - + z_0 a_2 \\
&\vdots \\
b_{n-1} &= a_{n-2} + z_0 a_{n-1} \\
b_n &= a_{n-1}
\end{aligned}$$

Note that b_n is known, since a_{n-1} is known. Working backwards from this line by line, we obtain $p(z_0)$. Notice that there is a total of n multiplications needed to do so. Contrast this with the $1 + 2 + \ldots + n + 1 = O(n^2)$ multiplications needed when evaluating $p(z_0)$ by computing $a_2 z^2, a_3 z^3$ and so on in succession.

6 Interpolation

While finding a least fit line to a series of data points (see the least squares minimization problem in these notes) makes sense if the data points do indeed have a linear relation to one another, it is grossly inaccurate otherwise. One way around this is to find nonlinear functions that interpolate the data. If we assume these functions are analytic, then we end up with a possibly infinite polynomial interpolating the data in question. We now direct our attention to deriving the appropriate interpolating polynomial, given a set of associations $\{(x_i, y_i\}_{i=0}^n$, where $f(x_i) = y_i$. We shall first establish a lemma.

Lemma 7 (Vandermonde Determinant)**.** *Let*

$$V_{n-k} \doteq \begin{bmatrix} 1 & x_k & x_k^2 & \dots & x_k^n \\ 1 & x_{k+1} & x_{k+1}^2 & \dots & x_{k+1}^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}$$

Then

$$\det V_{n-k} = \prod_{k<i<j<n} (x_i - x_j)$$

Proof. We first transform the first row and column to e_1 via row and column operations. More precisely, we multiply the n column by $-x_0$ and then add to the $n+1$ column, then multiply the $n-1$ column by x_0 and add to the n column, all the way down to the first column. This gives

$$V'_{n-k} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & (x_{k+1} - x_k) & x_{k+1}(x_{k+1} - x_k) & x_{k+1}^2(x_{k+1} - x_k) & \dots & x_{k+1}^{n-1}(x_{k+1} - x_k) \\ 1 & (x_{k+2} - x_k) & x_{k+2}(x_{k+2} - x_k) & x_{k+2}^2(x_{k+2} - x_k) & \dots & x_{k+2}^{n-1}(x_{k+2} - x_k) \\ \vdots & & & & & \\ 1 & (x_n - x_k) & x_n(x_n - x_k) & x_n^2(x_n - x_k) & \dots & x_n^{n-1}(x_n - x_k) \end{bmatrix}$$

Now, multiplying the first row by -1 and adding it to the remaining rows, we obtain

$$V'_{n-k} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & (x_{k+1} - x_k) & x_{k+1}(x_{k+1} - x_k) & x_{k+1}^2(x_{k+1} - x_k) & \dots & x_{k+1}^{n-1}(x_{k+1} - x_k) \\ 0 & (x_{k+2} - x_k) & x_{k+2}(x_{k+2} - x_k) & x_{k+2}^2(x_{k+2} - x_k) & \dots & x_{k+2}^{n-1}(x_{k+2} - x_k) \\ \vdots & & & & & \\ 0 & (x_n - x_k) & x_n(x_n - x_k) & x_n^2(x_n - x_k) & \dots & x_n^{n-1}(x_n - x_k) \end{bmatrix}$$

Observe that factoring $x_{k+i} - x_k$ from the $i+1$ row, for each $1 \leq i \leq n$, we obtain

$$\begin{bmatrix} 1 & 0 \\ 0 & V_{n-k-1} \end{bmatrix}$$

Hence, the problem of computing the Vandermonde matrix determinant is inherently recursive. Recall that multiplying rows and columns by non-zero constants and adding them to

each other does not impact the determinant of a matrix. However, if $A \mapsto A_c$ via multiplication of a row by some nonzero constant c, then $|A_c| = c|A|$. Hence, from our above computations and a cofactor expansion of V_n', it follows that

$$|V_{n-k}| = \prod_{k<i\leq n} (x_i - x_k)|V_{n-k-1}|$$
$$|V_2| = \left| \begin{bmatrix} 1 & x_{n-1} \\ 1 & x_n \end{bmatrix} \right| = x_n - x_{n-1}$$

which gives

$$\begin{aligned} |V_{n-k}| &= \prod_{k=0}^{n-2} \prod_{1\leq i\leq n} (x_i - x_k) \\ &= \prod_{k<i<j\leq n} (x_i - x_k) \end{aligned}$$

completing the proof. □

Theorem 8. *For an arbitrary data set $\{(x_i, y_i)\}_{i=0}^n$ with the x_i distinct, there exists a unique polynomial p of degree at most n such that*

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

Proof. For the existence and uniqueness, we write an ansatz of degree at most n

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

and try to find the a_i. Our problem reduces to solving

$$V_n a^T = y^T$$

where $\vec{a} = (a_0, a_1, \ldots, a_n)$, $\vec{y} = (y_0, y_1, \ldots, y_n)$, and where V_n is the $(n+1) \times (n+1)$ Vandermonde matrix

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix}$$

which is invertible by Lemma 7. Hence, the a_i are uniquely determined. □

Remark. Observe that we can find infinitely many polynomials of degree $m > n$ interpolating a given data set $\{(x_i, y_i)\}_{i=0}^n$. To see this, note that we can simply extend this data set to $\{(x_1, y_i\}_{i=0}^m$, where (x_i, y_i), $n < i < m$ are chosen arbitrarily with the x_i distinct, and then compute the inverse of the resulting Vandermonde matrix.

6.1 Newton Interpolation

As we have seen, finding exact solutions to $n \times n$ systems of form $Ax = b$ takes $O(n^3)$ computations in general. Our advantage for our linear system is that we are dealing with polynomials, which have an incredible amount of structure. Hence, we are motivated to utilize this to develop an $O(n)$ algorithm for finding the explicit form of our polynomial. To do so, we implement an idea of Newton, rewriting our polynomial as

$$\begin{aligned} p(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ &= \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1}(x - x_j), \quad \prod_{j=0}^{-1}(x - x_j) \doteq 1. \end{aligned}$$

The c_i can be computed in a naive fashion by plugging in x_i and solving for c_i, for each i. Computation of c_i will then depend on computations of all $c_{<i}$. Let us formalize all this.

Definition. Let f be a function which we wish to approximate via a polynomial, and $\{x_i, y_i\}_{i=0}^n$ a given data set with the x_i distinct. Let

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \ldots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

be the unique polynomial of at most degree n interpolating f on the given data set. We call

$$f[x_0, x_1, \ldots, x_n] \doteq c_n$$

a *divided difference* of f. It is the coefficient of the highest degree term in the polynomial of at most degree n interpolating the given data set.

Theorem 9 (Divided Differences)**.** *We have the recursive formula*

$$f[x_0, x_1, x_2, \ldots, x_n] = \frac{f[x_0, x_1, \ldots, x_{n-1}] - f[x_1, x_2, \ldots, x_n]}{x_n - x_0}$$

Proof. Let $p_{0,n}$ denote the unique polynomial of degree at most n that interpolates f at $x_0, x_1, \ldots, x_n$, and We define $p_{1,n}$ the unique polynomial of degree at most $n-1$ that interpolates f at $x_0, x_1, \ldots, x_n$. Then it is easy to check that

$$p_{0,n} = p_{1,n} + \frac{x - x_n}{x_n - x_0}[p_{1,n} - p_{0,n-1}]$$

Observe that the coefficient of x^n is given by the coefficient of x_n in the expression

$$\frac{x(p_{1,n} - p_{0,n-1})}{x_n - x_0}$$

completing the proof. □

6.2 Lagrange Interpolation

Given the data set $\{(x_i, y_i)\}_{i=0}^n$, write

$$p(x) = \ell_1(x)y_0 + \ell_2 y_1 + \ldots + \ell_n y_n$$

where the ℓ_i are polynomials of degree at most n. Then for $p(x)$ to interpolate this data, we must have

$$y_i(x) = \begin{cases} 1, & x = x_i \\ 0, & x \neq x_i \end{cases}$$

It is easy to check that

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

Definition. The polynomial $p(x) = \ell_1(x)y_0 + \ell_2 y_1 + \ldots + \ell_n y_n$ is called a *Lagrange polynomial*.

6.3 Hermite

Let $f(x) \in C^m(\mathbb{R})$ with

$$f^i(x_j) = b_{ij}, \qquad 0 \leq j \leq k,\ 0 \leq i \leq n_j \tag{6.1}$$

where b_{jk} is a constant. Can we find a polynomial $p(x)$ such that

$$p^i(x_j) = b_{ij}, \qquad 0 \leq j \leq k,\ 0 \leq i \leq n_j, \tag{6.2}$$

holds? If so, we call $p(x)$ a *Hermite polynomial*, and we say that $p(x)$ *interpolates* $f(x)$ at the *nodes* x_k.

Theorem 10. *If $\sum_{j=0}^k (n_j + 1) = m$, then there exists a unique polynomial of degree less than or equal to m satisfying (6.2).*

Proof. Write

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$$

where the constants $\{a_i\}_{0 \leq i \leq m}$ are to be determined. We wish to solve the system given by

$$\begin{gathered} a_0 + a_1 x_j + a_2 x_j^2 + \cdots + a_m x_j^m = b_{0j} \\ a_1 + 2a_2 x_j + 3a_3 x_j^2 + \cdots + m a_m x_j^{m-1} = b_{1j} \\ \vdots \\ n_j! a_{n_j} x_j^{m-n_j} + (n_j+1)! a_{n_j+1} x_j^{m-n_j-1} + \cdots + \frac{m!}{(m-n_j)!} a_m x_j = b_{n_j j} \end{gathered}$$

ranging over $0 \leq j \leq k$. In matrix form, this is expressed by $A\vec{a} = B$, where

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{bmatrix}, \; A_j = \begin{bmatrix} 1 & x_j & x_j^2 & \cdots & \cdots & \cdots & \cdots & x_j^m \\ 0 & 1 & 2x_j & \cdots & \cdots & \cdots & \cdots & mx_j^{m-1} \\ 0 & 0 & 2 & \cdots & \cdots & \cdots & \cdots & m(m-1)x_j^{m-2} \\ \vdots & \vdots & \vdots & \ddots & & & & \\ 0 & 0 & 0 & \cdots & n_j! & (n_j+1)!x_j & \cdots & \frac{m!}{(m-n_j)!}x_j^{m-n_j} \end{bmatrix},$$

$$\vec{a} = \begin{bmatrix} a_0 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}$$

$$B = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_k \end{bmatrix}, \; B_j = \begin{bmatrix} b_{0j} \\ b_{1j} \\ \vdots \\ b_{n_j j}. \end{bmatrix}$$

Note that A_j is a $(n_j + 1) \times (m + 1)$ matrix. Hence, for A to be a square matrix, we must have $\sum_{j=0}^{k}(n_j + 1) = m$.

Therefore, assuming this condition, to complete the proof it will be enough to show that A is invertible, or, equivalently, that if $A\vec{a} = \vec{0}$, then $\vec{a} = \vec{0}$. Proceeding, suppose $A\vec{a} = \vec{0}$. Then $\vec{a}$ defines a polynomial $p(x)$ with root x_j of multiplicity $n_j + 1$. Hence,

$$p(x) = q(x)\prod_{j=0}^{k}(x - x_j)^{n_j+1}.$$

However, note that $\prod_{j=0}^{k}(x - x_j)^{n_j+1}$ is of order $m + 1$, whereas the coefficients of $\vec{a}$ define $p(x)$ to be of order at most m. It follows that $q \equiv 0$. □

Remark. Observe that if $\sum_{j=0}^{k}(n_j + 1) < m$, then the system is underdetermined. This has at least one solution: create "extra" data such that $\sum_{j=0}^{k}(n_j + 1) = m$. Recall that undetermined systems have either no solutions, or infinitely many. We conclude that there are infinitely many polynomials of degree m which interpolate the given data.

6.4 Cubic Splines

The problem with Newton or Hermite interpolation is that if we have many data points, our interpolating polynomial p will have degree $n \gg 1$. Evaluating this polynomial at a point x_0 will then take $O(n)$ computations. To sidestep this problem, we seek to divide our data up into pieces and to interpolate each piece individually via a cubic polynomial. Once we have done this, we will then stitch these polynomials together.

Definition. A *spline* on an interval $[a, b]$ are polynomial pieces defined on disjoint subintervals $[a_j, b_j]$ which cover $[a, b]$.

Let S be the cubic spline that we seek to construct. We make the following assumptions.

1. On each interval $[t_{i-1}, t_i]$, S is a polynomial of degree at most 3.
2. S is C^2 on the boundary of each sub-interval.

By the first assumption, S'' is linear on each sub-interval. Hence, if $S_i''(t_i) = z_i$ and $S_i''(t_{i+1}) = z_{i+1}$, then

$$S_i''(x) = \frac{z_i}{h_i}(t_{i+1} - x) + \frac{z_{i+1}}{h_i}(x - t_i), \quad h_i \doteq t_{i+1} - t_i$$

Integrating this, we obtain an at most cubic polynomial with two unknown constants of integration C, D, which must be found along with the z_i. However, our regularity assumptions on the spline are powerful enough to allow us to do so.

7 Numerical Differentiation

Given a data set $\{x_i, y_i\}$, where $f(x_i) = y_i$, we can estimate f at a point z_i by approximating it via a linearization. More precisely, if $f \in C^2(I)$, where $\{x_i\} \subset I$, we write $f(x + h) = f(x) + hf'(x) + h^2 f''(x)/2$ and so

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} - hf''(\xi)/2 \\ &\approx \frac{f(x+h) - f(x)}{h} \end{aligned}$$

However, observe that as h becomes small, we begin to subtract *nearly equal quantities*. This results in a loss of precision. Worse, this error from loss of precision is magnified by $1/h$. To offset this error, we require a better approximation to $f'(x)$ than the $O(h)$ approximation we have produced above.

7.1 Richardson Extrapolation

The key to improving our approximation is to assume more regularity for f. We develop our improved approximation by first assuming f is analytic; we leave as an exercise how to adapt the argument for $f \in C^k$, but not necessarily analytic. Write

$$\begin{aligned} f(x+h) &= \sum_{k=0}^{\infty} h^k f^{(k)}(x)/k! \\ f(x+h) &= \sum_{k=0}^{\infty} (-1)^k h^k f^{(k)}(x)/k! \end{aligned}$$

Subtracting the second equation from the first and rearranging terms, we obtain

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \left[h^2 f^{(3)}(x)/3! + h^4 f^{(5)}(x)/5! + \ldots\right]$$

which we rewrite as

$$L(h) \doteq \phi(h) + a_2 h^2 + a_4 h^4 + \dots$$

The approximation $L \approx \phi(x)$ has $O(h^2)$ error. To obtain greater accuracy for an approximation to L, we seek to eliminate the h^2 term. Observe that $L(h) = L(h/c)$ for any nonzero constant c. Then one can check that

$$\begin{aligned} 3L(\cdot) = 4L(h/2) - L(h) &= [4\phi(h/2) + a_2 h^2 + a_4 h^4/4 + \dots] - [\phi(h) + a_2 h^2 + a_4 h^4 + \dots] \\ &= 4\phi(h/2) - \phi(h) - 3a_4 h^4/4 - 15a_6 h^6/16 - \dots \end{aligned}$$

and so

$$L = \frac{4}{3}\phi(h/2) - \frac{1}{3}\phi(h) - \frac{a_4}{4}h^4 - \dots$$

Hence, $L \approx \frac{4}{3}\phi(h/2) - \frac{1}{3}\phi(h)$ is a $O(h^4)$ approximation.

We can generalize this technique, known as *Richardson extrapolation*, to give an approximation to L of $O(h^{2j})$, for any desired $j \in \mathbb{N}$.

Exercise. Show that if $f \in C^k$ but $f \notin C^{k+1}$, then we can modify the Richardson extrapolation algorithm developed above to obtain an $O(h^{2j})$ approximation to L, for any positive integer $j < k$.

7.2 Numerical Differentiation Using Polynomial Interpolation

We use Newton interpolation, Lagrange interpolation, or splines to interpolate a given data set. Once the interpolating polynomial is obtained, we simply differentiate it to obtain our approximation to $f'(x)$. This has the advantage that we do not suffer loss of precision due to subtraction of nearly equal quantities, as we do when computing $f'(x)$ via linearization. However, constructing a polynomial p via interpolation, and then computing $p'(x)$ for various inputs x is computationally more costly.

8 Numerical Integration

Consider the data set $\{x_i, y_i\}_{i=1}^n$, where $x_i = a$, $x_n = b$, $f(x_i) = y_i$, and where we assume without loss of generality that $\{x_i\}$ is an increasing set. We can estimate the integral of f in the region $I \subset [x_1, x_n]$ using basic Calculus tools, such as right rectangles, left tangles, and trapezoids. Observe, however, that all these approaches are produced via linear splines between our data points.

We wish to generalize this approach to obtain more sophisticated, and precise, numerical integrals. To do so, we can either look for a polynomial or a spline that interpolates the given data points, and use this to find our approximation to the integral of f. More precisely, we have

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx$$

$$|E_\ell(b-a)| \le |\int_a^b f(x)\,dx| \le |E_u(b-a)|$$

where E_ℓ and E_u are lower and upper bounds, respectively, for the L^∞ error of our polynomial approximation $p(x)$. Hence, lower and upper bounds for the error of our approximation $p(x)$ are translated into lower and upper bounds for our integral approximation.

Using Lagrange interpolation, it is easy to see that

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^n f(x_i) \int_a^b l_i(x)\,dx \doteq \sum_{i=0}^n A_i f(x_i)$$

If the nodes are equally spaced, this is called a *Newton-Coates Formula*.

Exercise. Show that for uniform spacing $h = (b-a)/n$ and $x_i = a + ih$, the Newton-Coates formula produces the composite trapezoid formula

$$\int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{i=1}^n (x_i - x_{i-1})[f(x_{i-1}) - f(x_i)]$$

Exercise. Derive Simpson's rule

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\,[f(a) + 4f((a+b)/2) + f(b)]$$

via the Newton-Coates formula. The formula will guarantee that Simpson's rule is exact for polynomials of degree ≤ 2. Show that it is, in fact, exact for polynomials of degree ≤ 3.

Aside from integrating the Lagrange polynomials, another way we can compute the A_i is via the method of undetermined coefficients. More precisely, suppose we seek A_0, A_1, A_2 such that

$$\int_0^1 f(x)\,dx \approx A_0 f(0) + A_1 f(1/2) + A_2 f(1).$$

One way is to construct three equations for these three unknowns. Letting $f = 1$, $f = x$, and $f = x^2$, respectively, we compute

$$\begin{aligned} 1 &= A_0 + A_1 + A_2 \\ 1/2 &= 1/2A_1 + A_2 \\ 1/3 &= 1/4A_1 + A_2 \end{aligned}$$

from which we obtain $A_0 = 1/6$, $A_1 = 2/3$, and $A_2 = 1/6$. Since $\{1, x, x^2\}$ is a basis for the vector space of polynomials of degree ≤ 2, and since integration is a linear operation, we see that our approximation is in fact exact for all polynomials $f(x)$ of degree ≤ 2.

8.1 Gaussian Quadrature

Given a data set $\{(x_i, y_i)\}_{i=0}^n$, the Newton-Coates formula

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^n f(x_i) \int_a^b l_i(x)\,dx \doteq \sum_{i=0}^n A_i f(x_i)$$

is exact for polynomials of degree $\leq n$, by construction. However, not all data is created equal: some data "represents" f better than others. This is reflected in the following marvelous theorem of Gauss.

Theorem 11. *Let q be a nonzero polynomial of degree $n+1$ that is $L^2[a,b]$ orthogonal to the space of polynomials Π_n of degree n. That is, for all $p \in \Pi_n$*

$$\langle p, q \rangle_{L^2[a,b]} = \int_a^b p\bar{q}\, dx = 0.$$

Then if we let the $\{x_i\}_{i=0}^n$ be the $n+1$ roots of q, the Newton-Coates formula

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

is exact for all polynomials of degree $\leq 2n+1$.

Proof. Let $p(x)$ be a polynomial of degree $\leq 2n+1$, and let q be of degree $n+1$ and $L^2[a,b]$ orthogonal to Π_n. Dividing p by q, we obtain

$$p = qs + r$$

Then

$$\int_a^b p\, dx = \int_a^b qs\, dx + \int_a^b r\, dx = \int_a^b r\, dx = \sum_{i=0}^n A_i r(x_i)$$

where the last two equalities follow from orthogonality and the fact that Newton-Coates is exact for polynomials of degree $\leq n$. Lastly, we observe that since the x_i are roots of q, we have

$$\sum_{i=0}^n A_i r(x_i) = \sum_{i=0}^n A_i p(x_i)$$

which completes the proof. □

9 Finite Difference Method for Differential Equations

9.1 Burgers Equation

Recall the Burgers initial value problem (ivp)

$$u_t = \frac{1}{2}(u^2)_x, \tag{9.1}$$

$$u(x,0) = u_0(x), \qquad x, t \in \mathbb{R}. \tag{9.2}$$

Note that for small *stepsize* (or *mesh*) $h, k > 0$,

$$u_t(x,t) \approx \frac{u(x,t+k) - u(x,t)}{k},$$
$$u_x(x,t) \approx \frac{u(x+h,t) - u(x,t)}{h}.$$

Set

$$x_i = ih, \quad i \in \mathbb{Z}$$
$$t_j = jk, \quad j \in \mathbb{N}$$

and let

$$u_{i,j} = u(x_i, t_j).$$

Then the discretized Burgers ivp takes the form

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{1}{2h}\left(u_{i+1,j}^2 - u_{i,j}^2\right),$$
$$u_{i,0} = u_0(ih)$$

or

$$u_{i,j+1} = u_{i,j} + \frac{k}{2h}\left(u_{i+1,j}^2 - u_{i,j}^2\right), \tag{9.3}$$
$$u_{i,0} = u_0(ih). \tag{9.4}$$

Note that (9.3)-(9.4) gives us an explicit numerical solution to the Burgers ivp (9.1)-(9.2). To illustrate this, we set $j = 0$, and obtain

$$u_{i,1} = \underbrace{u_{i,0} + \frac{k}{2h}\left(u_{i+1,0}^2 - u_{i,0}^2\right)}_{\text{known from initial data (9.4)}}. \tag{9.5}$$

Similarly, for $j = 1$, we have

$$u_{i,2} = \underbrace{u_{i,1}}_{\text{known from (9.5)}} + \frac{k}{2h}\left(\underbrace{u_{i+1,1}^2}_{\text{known from (9.5)}} - \underbrace{u_{i,1}^2}_{\text{known from (9.5)}}\right).$$

This process can be continued indefinitely to find $u_{i,j}$ for any $j \in \mathbb{N}$, and is called an *explicit finite difference method* for numerically solving the Burgers ivp. An important drawback to this method is that it converges very slowly to solutions of the Burgers ivp.

10 Well-Conditioned and Ill-Conditioned Linear Problems

Now that we have developed techniques for solving $n \times n$ systems $Ax = b$, we'd like to understand how sensitive our system is to perturbations. More precisely, we can ask the following two questions:

a) If we shift $x \mapsto \tilde{x}$, how much does $b \mapsto \tilde{b}$ shift, relative to the size of b?

b) If we shift $b \to \tilde{b}$, how much does the solution $x \mapsto \tilde{x}$ shift, relative to the size of x?

Definition. Let A be a square, invertible matrix. We call $\kappa(A) \doteq \|A\|\|A^{-1}\|$ its *condition number.*

Theorem 12. *We have the estimates*

$$\kappa(A)^{-1}\frac{\|b-\tilde{b}\|}{\|b\|} \leq \frac{\|x-\tilde{x}\|}{\|x\|} \leq \kappa(A)\frac{\|b-\tilde{b}\|}{\|b\|}.$$

Proof. Observe that

$$\|b-\tilde{b}\|\|x\| = \|A(x-\tilde{x})\|\|A^{-1}b\| \leq \kappa(A)\|x-\tilde{x}\|\|b\|$$

and

$$\|x-\tilde{x}\|\|b\| = \|A^{-1}(b-\tilde{b})\|\|Ax\| \leq \kappa(A)\|b-\tilde{b}\|\|x\|$$

Rearranging the terms in each expression, we obtain the lower and upper bounds for the size of the relative shift of $\tilde{x}$ from x, completing the proof. □

Hence, large condition numbers imply that small perturbations in b can have large impacts on the resulting perturbation of x, and vice versa. Motivated by this, we have the following definitions.

Definition. For a square matrix A, if $\kappa(A) \leq 1$, we say A is well-conditioned. Otherwise, we say A is ill-conditioned.

Remark. The upper-bound we use for the upper bound necessary for A to be well-conditioned is somewhat arbitrary, and ultimately depends on the problem we are analyzing and how accurate our collected data b is. If we are dubious as to the accuracy of our data-collecting techniques, we will need $\kappa(A)$ to be small, perhaps much smaller than just 1, in order to ensure that the x we come up with as solution to $Ax = b$ is not far from the actual x occurring in the real-world. In this case we would define a well-conditioned matrix as having a condition number less than some c, where $c \ll 1$.

Remark. The condition number can also be thought of as a measure of how close a matrix is to not being invertible. For example, consider

$$A_\varepsilon = \begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix}$$

A simple computation shows that $\kappa(A_\varepsilon) = O(1/\varepsilon^2)$. Hence, $\kappa(A_\varepsilon) \to \infty$ as $\varepsilon \to 0$. Of course, the matrix

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \lim_{\varepsilon \to 0} A_\varepsilon$$

is not invertible.

11 Least Squares Problem

11.1 Orthogonal Transformations and Hilbert Spaces

We recall that linear transformations on finite-dimensional spaces are a combination of scalings, translations, and rotations. Consequently, since matrices are linear transformations, we would like to factor our matrices into a product of rotation matrices, and scaling matrices. However, we must first formalize our notion of rotation.

Definition. A Hilbert Space V is a vector space equipped with a conjugate bilinear form $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{C}$. More precisely, we have

a) Symmetry: $\langle x, y \rangle = \overline{\langle y, x \rangle}$.

b) Linearity: $\langle \lambda x + y, z \rangle = \lambda \langle x, z \rangle + \langle y, z \rangle$.

c) Reflexivity: $\langle x, x \rangle$ is real and positive, and $\langle x, x \rangle = 0$ if and only if $x = 0$.

It is easy to check that the relation

$$\|v\| = \sqrt{\langle v, v \rangle}$$

defines a norm on any inner product space. If $\langle x, y \rangle$ is real valued, we can think of it as the size of the angle between x and y. Indeed, in $\mathbb{R}^2$ equipped with the standard euclidean inner product, one can check that

$$\langle x, y \rangle = \|x\| \|y\| \cos \theta$$

Hilbert spaces have other important properties. We shall utilize the next one frequently.

Theorem 13. *Let H be a Hilbert space, and $A : H \to H$ a bounded linear operator. Then there exists a map $A^\star : H \to H$ with the property that*

$$\langle Au, v \rangle = \langle u, A^\star v \rangle$$

for any $u, v \in H$. We call $A^\star$ the adjoint *of A.*

11.2 Simplification of Least Squares Problem via Adjoints

Ideally, we wish to find exact solutions to the linear problem $Ax = b$, where A is an $m \times n$ complex matrix, x is an $n \times 1$ column vector, and b is an $m \times 1$ column vector. However, this is not always possible. For example, if rank A $< m$, then the column space of A may not span b. In these situations, we look for $\tilde{x}$ such that $\|b - A\tilde{x}\|$ is minimized. The choice of norm is irrelevant in finite-dimensional vector spaces (a simple but useful corollary of Theorem 14), though the Euclidean norm (ℓ^2) is a powerful choice, since we can then equip our normed space with the inner product

$$\langle u, v \rangle \doteq \sum_i u_i \bar{v}_i$$

$$\langle u, u \rangle = \|u\|_{\ell^2}^2.$$

Recall that operators on inner-product spaces always posses an adjoint. We now make use of this important property to simplify our minimization problem considerably.

Theorem 14. *Let A be an $m \times n$ complex matrix with rank$(A) = n$. If x is a point such that $A^\star(Ax - b) = 0$, then x is the unique solution to the least squares problem.*

Proof. Assume $A^\star(Ax - b) = 0$. We first prove that x must be a least squares solution. Observe that $\overline{A^\star(Ax - b)} = 0$, or $A^T\overline{Ax - b} = 0$. Hence, $Ax - b$ is orthogonal to the column space of A. Let y be any other point. Since $A(x - y)$ lives in the column space of A , we see that $\langle Ax - b, A(x - y)\rangle = 0$. Hence, by the Pythagorean theorem,

$$\|Ax - b\|_{\ell^2}^2 = \|Ay - b + A(x - y)\|_{\ell^2}^2 = \|Ay - b\|_{\ell^2}^2 + \|A(x - y)\|_{\ell^2}^2 \geq \|Ay - b\|_{\ell^2}^2.$$

Hence, x is a least squares solution. Assume y is a least squares solution. Then by our previous work, we must have $\|A(x - y)\|_{\ell^2} \neq 0$. This implies $x - y$ lives in the null space of A. But since rank$(A) = n$, by the rank-nullity theorem, the dimension of the null space is 0. We conclude that $x = y$. □

Observe that if A is unitary, we obtain x immediately:

$$0 = A^\star(Ax - b) = x - A^\star b \implies x = A^\star b.$$

This is a simplistic example that illustrates that unitary matrices simplify our least squares computations considerably. Hence, it makes sense to look at decompositions of A utilizing unitary matrices, in an attempt to simplify equations of form $A^\star(Ax - b = 0)$ even further. More generally, suppose we can find a decomposition $A = QR$, where Q is $m \times m$ unitary, and R is $m \times n$ upper-triangular. Then

$$0 = A^\star(Ax - b) = R^\star Q^\star(Ax - b) = R^\star Q^\star(QRx - b) \implies R^\star Rx = R^\star Q^\star b.$$

Hence, the QR decomposition permits us to convert our problem of finding a least square into the equivalent problem of finding a solution x to a problem of LU type, which is much more tractable.

We now discuss the two most popular QR decomposition algorithms.

Gram-Schmidt

Theorem 15. *Let $\{x_1, x_2, \ldots, x_n\}$ be a linearly independent in C^n. Then the set $\{u_1, u_2, \ldots, u_n\}$ is an orthonormal basis for C^n, where*

$$u_k = \frac{x_k - \sum_{i<k}\langle x_k, u_i\rangle u_i}{\|x_k - \sum_{i<k}\langle x_k, u_i\rangle u_i\|_{\ell^2}}$$

Proof. It is a straightforward computation of $\langle u_i, u_j\rangle$. □

Applying the Gram-Schmidt process to the columns of a matrix A, we obtain the decomposition BT, where B is an $m \times n$ matrix consisting of the orthonormalized column vectors of A, and T consists of an $n \times n$ upper triangular matrix with positive diagonal, whose entries come from saved computations in the Gram-Schmidt algorithm.

Substituting this decomposition into $A^\star(Ax - b) = 0$, we

$$T^\star Tx = T^\star B^\star b$$

which is again a problem of LU type.

Remark. Instead of resorting to Gram-Schmidt or some other factorization, we could attempt to solve $A^\star(Ax-b)=0$ directly. That is, we attempt to solve $A^\star Ax = A^\star b$ via a Cholesky factorization ($A^\star A$ is Hermitian and positive semidefinite). The problem with this approach is that we may have $\kappa(A^\star A) >> k(A)$. A good example of this is

$$A = \begin{bmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{bmatrix}$$

Householder

A Householder decomposition is of form $A = QR$, where Q is a $m \times m$ unitary matrix, and R is an $m \times n$ upper triangular matrix. Observe that the columns of a unitary matrix are orthonormal (they have other useful properties), so this decomposition has better properties in general than the Gram-Schmidt decomposition. Indeed, substituting into $A^\star(Ax-b)$, we obtain the LU type problem

$$R^\star Rx = (QR)^\star b.$$

Singular Value Decompositions

We now discuss arguably the most useful decomposition of an arbitrary $m \times n$ matrix. We first require a lemma.

Lemma 16. *A Hermitian, positive semidefinite matrix has real positive eigenvalues.*

Proof. We first show that any Hermitian matrix has real eigenvalues. Suppose A is Hermitian. Then

$$v^\star A^\star v = (Av)^\star v = \bar{\lambda}\|v\|_{\ell^2}^2$$

and

$$v^\star Av = v^\star \lambda v = \lambda \|v\|_{\ell^2}^2.$$

Since A is Hermitian, $A = A^\star$, so we must have $\bar{\lambda} = \lambda$, which implies that λ is real. Furthermore, since A is positive definite, it follows that $v^\star Av \geq 0$, and hence, $\lambda \geq 0$. □

Theorem 17 (Singular Value Decomposition)**.** *Let A be an arbitrary $m \times n$ matrix with complex entries. Then there exists a decomposition $A = PDQ$, where P is $m \times m$ unitary, D is a $m \times n$ diagonal matrix, and Q is a $n \times n$ unitary matrix. This decomposition is not necessarily unique.*

Proof. The non-uniqueness of the PDQ decomposition is easy: the 0 matrix has infinitely many PDQ type decompositions (just take $D = 0$, and P, Q to be arbitrary unitary matrices).

To prove existence of the PDQ decomposition for arbitrary $m \times n$ matrices A, we first assume without loss of generality that $m \geq n$ (otherwise, we consider A^T). By Lemma 16, for each of n linearly independent eigenvectors u_i of $A^\star A$, there exist associated positive eigenvalues σ_i^2. Furthermore,

$$\|Au_i\|_{\ell^2}^2 = \langle Au_i, Au_i \rangle = \langle u_i, A^\star Au_i \rangle = \sigma_i^2 u_i \tag{11.1}$$

We can also choose these eigenvectors such that they are orthonormal. To see this, recall that $A^\star A$ is square, and that square matrices are unitarily similar to upper-triangular matrices (in fact, $A^\star A$ is Hermitian, so it is unitarily similar to a diagonal matrix). Since square upper-triangular matrices have eigenvectors e_i, the eigenvectors of a square matrix are given by a unitary transformation of the e_i, which preserves their orthonormality since

$$\langle Qe_i, Qe_j\rangle = \langle e_i, Q^\star Qe_j\rangle = \langle e_i, e_j\rangle = \delta_{ij}.$$

Hence, we have the decomposition $A^\star A = \tilde{D}^2 Q$ where $\tilde{D}$ is an $n \times n$ matrix consisting of the positive square roots of eigenvalues σ_i^2 of $A^\star A$, and Q is the $n \times n$ unitary matrix with rows $u_i^\star$.

We now massage this decomposition to come up with PDQ. In effect, we seek to factor $A^\star \tilde{D} = PD$, where D is a $m \times n$ diagonal matrix with diagonal consisting of the $|\sigma_i|$, and P is $m \times m$ unitary. Assume non-zero eigenvalues for $i \leq r$, where $r \leq n$, and zero otherwise. Then by (11.1), $Au_i \neq 0$ for $i \leq r$, and $Au_i = 0$ otherwise. Set $v_i = Au_i/\sigma_i, 1 \leq i \leq r$. It is easy to check that these v_i are orthonormal. Using Gram-Schmidt, extend the set $v_1, \ldots, v_r$ to an orthonormal set $\{v_1, \ldots, v_m\}$, and set P to be the matrix with the v_i on its columns. Then one can verify that $A = PDQ$. Notice that this decomposition is not unique: it depends on how we arrange our orthonormal vectors in P and Q. □

Substituting a singular value decomposition into our minimization result $A^\star(Ax - b) = 0$, we obtain

$$0 = Q^\star DP^\star(PDQx - b) = Q^\star D^2 Qx - Q^\star DP^\star b \implies D^2 Qx = DP^\star b$$

Let D^+ be the matrix with $1/\sigma_i$ in its diagonal for $\sigma_i \neq$, and $0's$ otherwise. This is what's known as a *pseudo-inverse* of D. Multiplying both sides of the last expression by $Q^\star(D^+)^2$, we obtain

$$x = Q^\star D^+ P^\star b$$

Hence, we have proved the following result.

Theorem 18. *Let $Ax = b$ be a system to which we seek least-squares solution, where A has singular value decomposition PDQ. Then the solution is given by $x = A^+ b$, where $A^+ = Q^\star D^+ P^\star$*

This motivates the following definition.

Definition. Let A be an arbitrary $m \times n$ matrix with singular value decomposition PDQ. Then we call $A^+ = Q^\star D^+ P^\star$ its *pseudoinverse*.

It can be shown that even though singular value decompositions for a matrix A are not unique, the pseudoinverse A^+ is.

12 Finding Eigenvectors and Eigenvalues

Having seen the importance of the singular value decomposition in the preceding section, we would like to develop an algorithm for finding the eigenvalues and corresponding eigenvectors of $A^\star A$ for any given matrix A. From there, we saw in the last section that we can orthonormalize the set of eigenvectors via Gram-Schmidt and, if necessary, extend this set to form a basis for C^n. Once we do that, our PDQ decomposition will be complete.

12.1 The Power Method

We now develop an algorithm for finding the eigenvalue of maximum modulus for a non-zero $n \times n$ matrix A and its corresponding eigenvector. We require that there is a unique eigenvalue of maximum modulus, and that A be diagonalizable. Proceeding, suppose λ_1 is the eigenvalue of maximum modulus. If $\lambda_1 = 0$, then $A = 0$, contradicting our earlier assumption. Hence, $\lambda_1 \neq 0$, with corresponding eigenvector u_1. Choose $x^{(0)} \in \mathbb{C}^n$ that is n-dimensional. Then we must have

$$x^{(0)} = \sum_{i=1}^{n} a_i u_i, \quad a_i \neq 0 \ \forall i$$

It follows that

$$x^{(k)} = A^k x^{(0)} = \sum_{i=1}^{n} a_i \lambda_i^k u_i = \lambda_1^k \sum_{i=1}^{n} a_i (\lambda_i/\lambda_1)^k u_i = a_1 \lambda_1^k u_1 + \varepsilon_k$$

Let $\phi : \mathbb{C}^n \to \mathbb{C}$ be the projection function of an input onto its first coordinate. Then

$$\phi(x^{(k)})/\phi(x^{(k-1)}) \to \lambda_1$$

from which we obtain

$$x^{(k)}/\lambda_1^k \to a_1 u_1$$

which is the corresponding eigenvector. From a theoretical standpoint, using this for finding the corresponding eigenvector is perfectly fine. However, numerically it is a bit imprecise, since we are using an approximation to λ_1 (due to machine rounding and termination of our algorithm after a pre-specified number of iterations) and then taking powers of it to compute our eigenvector. There is a risk that this will compound the errors already present, introducing complexity to our problem. There is a simple way to bypass this: we can normalize $x^{(k)} \to x^{(k)}/\|x^{(k)}\| \doteq \tilde{x}^{(k)}$ at each step, and still obtain

$$\phi(\tilde{x}^{(k)})/\phi(\tilde{x}^{(k-1)}) \to \lambda_1.$$

However, now we also have

$$\tilde{x}^{(k)} = \frac{a_1 \lambda_1^k u_1 + \varepsilon_k}{\|a_1 \lambda_1^k u_1 + \varepsilon_k\|} \to u_1/\|u_1\|$$

which is an eigenvector corresponding to the eigenvalue λ_1.

12.2 Inverse Power Method

This is almost identical to the Power method, with the additionally caveat that we require A to be invertible. We observe that for square matrices, $Av = \lambda v$ implies $A^{-1} v \lambda^{-1} v$ so long as A is invertible or, equivalently, that $\lambda \neq 0$. Hence, if λ is the unique eigenvalue of minimum modulus of a diagonalizable matrix A, it is the unique eigenvalue of maximum modulus of the diagonalizable matrix A^{-1}. Hence, we can apply the power method on A^{-1} to find λ^{-1},

from which we recover λ. In practice, it is not computationally efficient to compute A^{-1} and then run the power method. Rather, we rewrite $x^{(k)} = (A^{-1})^k x^{(0)}$ in equivalent form as

$$x^{(k)} = Ax^{(k-1)}$$

We can now apply an LU decomposition to make it easy to compute $x^{(k)}$. To see that an invertible diagonalizable $n \times n$ matrix A has cofactors that are all zero (and hence, admits an LU decomposition), simply observe that $A = PDP^{-1}$ and where P has columns consisting of n linearly independent eigenvectors of A, and D has diagonal with no zeroes in it. The rest of the algorithm is then analogous to the power method.

12.3 Shifted Power Method

Lastly, we would like to find all eigenvalues between those with largest and smallest modulus, respectively. First, observe that if a random chosen μ is closest to a simple eigenvalue λ of a square matrix A, then $\lambda - \mu$ is the unique eigenvalue of minimum modulus of $A - \mu I$. If $A - \mu I$ is invertible, we can run the inverse power method to obtain $\lambda - \mu$, from which we are able to obtain λ.

13 Linear Iteration

All linear iterations are of form

$$x^{k+1} = Ax^k + b.$$

We would like to know under what conditions the iteration converges. To explore this line of questioning further, we will need a series of lemmas.

Lemma 19 (Schur's Lemma)**.** *Every square matrix is unitarily similar to an upper triangular matrix whose off diagonal entries are arbitrarily small.*

Corollary 20. *If A is an $n \times n$ matrix with complex entries, then*

$$\rho(A) = \inf_{\|\cdot\|} \|A\|$$

Proof. We shall show that $\rho(A) \leq \inf_{\|\cdot\|} \|A\|$ and $\rho(A) \geq \inf_{\|\cdot\|} \|A\|$, respectively. If A is the 0 matrix, there is nothing to prove. Otherwise, A has one nonzero eigenvalue, with a corresponding eigenvector which cannot be 0. Hence,

$$|\lambda| \cancel{\|x\|} = \|Ax\| \leq \|A\| \cancel{\|x\|}.$$

from which it follows that $\rho(A) \leq \|A\|$.

For the reverse direction, we first apply Schur's Lemma to decompose $SAS^{-1} = D + T_\varepsilon$. Observing that similar matrices have the same spectrum, we obtain

$$\|SAS^{-1}\|_{\ell_\infty} \leq \rho(A) + \varepsilon.$$

It is easy to check that, for fixed S, $\|A\|_{\ell^{\infty\prime}} \doteq \|SAS^{-1}\|_{\ell^\infty}$ is a norm. Since $\varepsilon > 0$ can be taken to be arbitrarily small, we conclude that $\rho(A) \geq \inf_{\|\cdot\|} \|A\|$. □

We now have the tools to prove the following.

Theorem 21. *The linear iteration $x^{k+1} = Ax^k + b$ converges for any initial vector x^0 if and only if $\rho(A) < 1$. If so, then $x^k \to (I - A)^{-1}b$.*

Proof. To prove sufficiency, we first observe that convergence implies a fixed point to the map $x \mapsto Ax + b \doteq Tx$. This map is a contraction if

$$\|Tx - Ty\| \leq \|x - y\|,$$

which simplifies via linearity of T to

$$\|A(x - y)\| \leq \|x - y\|.$$

But

$$\|A(x - y)\| \leq \|A\|\|x - y\|$$

If $\rho(A) < 1$, by Corollary 20 there exists a norm $\|\cdot\|$ such that $\|A\| < 1$. Hence, $T : C^n \to C^n$ is a contraction under this norm, and so Therefore, x^k converges to some x in this norm. Since all norms are equivalent on finite dimensional spaces, it follows that x^k converges to x in any norm. Since x is a fixed point of T, we must have $x = Ax + b$, or $x = (I - A)b$. To prove necessity, assume that $\rho(A) \geq 1$. Let $b = 0$ and x^0 be the eigenvector with largest eigenvalue λ amongst all eigenvalues. Then $|\lambda| \geq 1$, and so

$$\|x^{k+1} - x^k\| = \|(\lambda^k - \lambda^{k-1})x^0\| = |\lambda|^k\|(1/\lambda - 1)x^0\| \to \infty, \quad \lambda > 1$$

If $\lambda = 1$, then $-\lambda$ is also an eigenvalue, with corresponding eigenvector $-x^0$. Therefore, assuming without loss of generality that $\lambda = -1$, we obtain

$$\|x^{k+1} - x^k\| = \|(\lambda^k - \lambda^{k-1})x^0\| = 2\|x^0\|$$

which implies $\{x^k\}$ is not Cauchy. □

Corollary 22 (Neumann Iteration)**.** *The sum*

$$S \doteq \sum_{k=0}^{\infty} A^k$$

converges if and only if $\rho(A) < 1$. If so, then $S = (I - A)^{-1}$.

Proof. One can generalize the proof of Theorem 21 to let b and our initial choice x^0 be matrices. Choosing $b = 0$ and $x^0 = I$, we see that

$$\begin{aligned} x^1 &= I, \\ x^2 &= A + I, \\ x^3 &= A^2 + A + I, \\ &\vdots \\ x^k &= \sum_{i=0}^{k} A^i \end{aligned}$$

which by Theorem 21 converges to $(I - A)^{-1}$. □

Remark. This corollary is important in that it provides us an $O(n^3)$ iterative method (multiplication of two $n \times n$ matrices takes $O(n^3)$ computations) for computing the inverses of matrices. More precisely, note that if $\rho(I+A) \geq 1$, then $\rho(I+cA) < 1$ for some $-1 < c < 1$. Hence, we can apply Neumann iteration to $I + cA$ to recover the inverse of cA. From there, it is easy to find the inverse of A.

13.1 Iterative Refinement Schemes

Observe that if $B = A^{-1}$, then

$$x = x + B(b - Ax).$$

If $B \approx A^{-1}$, then we set

$$x^{k+1} = x^k + B(b - Ax^k).$$

Let $Q = B^{-1}$. Then we obtain

$$x^{k+1} = Q^{-1}(Q - A)x^k + Q^{-1}b.$$

Definition. We call Q a *splitting* matrix. If $Q = I$, the iterative scheme is called *Richardson Iteration*. If Q is a diagonal matrix with diagonal equal to that of A, then it is called *Jacobi Iteration*. Lastly, if Q is a lower triangular matrix with lower triangular part (including the diagonal) equal to that of A, the iterative scheme is called *Gauss-Seidel Iteration*.

Notice that in all three cases, Q^{-1} is easily computed. Computing the exact inverse of an $n \times n$ matrix is generally an $O(n^3)$ computation, and computing an approximate inverse via Neumann iteration is also, in general, $O(n^3)$. For these special Q, it is $O(n)$ (the exact inverse is computed via backward substitution of n rows, while in the Neumann iteration we are multiplying by zero on $O(n^2)$ times) .

However, if we wish to bypass finding the inverse of Q, we can do so. More precisely, if $Q = I$, we obtain observe that

$$x^{k+1} = x^k + e^k, \quad e^k \doteq A^{-1}(b - Ax^k)$$

Then one way to run our iterative refinement as follows: Given base case x^0, we compute

i) $r^k = b - Ax^k$

ii) $Ae^k = r^k$

iii) $x^{k+1} = x^k + e^k$

With an appropriate chosen splitting matrix Q, the analogous iteration may become more computationally efficient. We then have

$$Qx^{k+1} = (Q - A)x^k + b.$$

Note that for Gauss-Seidel iteration, the choice of Q is clever, in that this iteration is converted to

$$Lx^{k+1} = -Ux^k + b$$

where L and U are the upper and lower triangular parts of A respectively, where U has zeroes on its diagonals.

Remark. Hence, it is quite clear that it is more efficient to choose a splitting matrix before iterating. However, one must be careful to first identify whether or not the Gauss-Seidel scheme converges for a given matrix A. If it does, then in general it converges faster in general than the Jacobi and Richardson iterative schemes. If it does not, then we must resort to using another scheme.

14 Finding Exact Solutions to Matrix Systems

We are interested in exact solutions to the system $Ax = b$, where A is an invertible $n \times n$ matrix. The unique solution is given by $x = A^{-1}b$; however, it is numerically inefficient to compute A^{-1} via Gaussian elimination. We have seen in the previous section that A^{-1} can be estimated to any degree of accuracy we wish via Neumann iteration. However, another, more common technique is to decompose A into a product of matrices that makes the system $Ax = b$ easy to solve.

14.1 LU Decompositions

Observe that if we can write $A = LU$, where L and U are lower and upper $n \times n$ matrices respectively, then solving $Ax = b$ reduces to solving $Ly = b$ and $Ux = y$. However, these two systems can easily be solved by backwards substitution and reverse backwards substitution, respectively.

Doolittle Factorization

Assuming that L has unit diagonal, and seek to find the remaining entries of L and U. Let L_i denote the ith row of L, and U^j denote the jth column of U. Observe that

$$a_{ij} = L_i U^j = \sum_{s=1}^{n} \ell_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj}.$$

Using this formula, we compute the first row of U the first column of L as follows. First we aim to compute the first row of U. To do so, we look at $L_1 U^k$, $1 \leq k \leq n$ and use the key fact that $\ell_{11} = 1$ and $\ell_{1i} = 0, 1 < i \leq n$ to compute $u_{11}, u_{12}, \ldots, u_{1n}$. Having computed the first row, we now compute the first column of L by looking at $L_i U^1$, $2 \leq i \leq n$, and using the fact that U is upper triangular. For the inductive step, assume that we have computed the first $k-1$ rows of U, and the first $k-1$ columns of L. Note that

$$a_{kk} = \sum_{s=1}^{k-1} l_{ks} u_{sk} + u_{kk}$$

which implies

$$u_{kk} = a_{kk} - \sum_{s=1}^{k-1} l_{ks} u_{sk}.$$

However, both terms on the right hand side are known. In particular, we know the second term via the inductive hypothesis. Hence, we have computed $u_k k$.

Now that we have computed u_{kk}, we use it as a pivot to compute the values in the kth column of L. Observe that for $k < i \leq n$

$$a_{ik} = L^i U^k = \sum_{s=1}^{\min(i,k)} \ell_{is} u_{sk} = \sum_{s=1}^{k-1} \ell_{is} u_{sk} + \ell_{ik} u_{kk}$$

We then solve for ℓ_{ik}. Note that every other in the expression on the right is known via the inductive hypothesis, and our previous computation of u_{kk}.

Now that we have computed the kth column of L, we use it to compute the kth row of U. Observe that for $k < i \leq n$

$$a_{ki} = L^k U^i = \sum_{s=1}^{\min(i,k)} \ell_{ks} u_{si} = \sum_{s=1}^{k-1} \ell_{ks} u_{si} + \ell_{kk} u_{ki}$$

Observe that u_{ki} is the only term on the right hand side that is not known. Every other term is known thanks to our computation of the kth column of L earlier, and our inductive hypothesis. Hence, we can solve for u_{ki}, so long as divisions by zero do not occur.

This completes the LU decomposition algorithm.

Definition. When L is assumed to have unit diagonal as above, we call the algorithm a *Doolittle factorization.* If U is assumed to have unit diagonal, it is called a *Crout factorization.* If A is symmetric and positive definite, it is called a *Cholesky factorization.*

Observe that we have glossed over the fact that the algorithm fails if divisions by 0 when solving for terms in L or U. Fortunately, we have the following.

Proposition 23. *Let A be an invertible $n \times n$ matrix. Then there exists a permutation of the rows P such that PA admits an LU decomposition. More precisely, no divisions by 0 will occur when running an LU decomposition algorithm on PA.*

Remark. Note that x is a solution to $Ax = b$ if and only if $PAx = Pb$.

Exercise. Show that a real, symmetric, positive definite matrix A has a unique factorization LL^T, where L is lower triangular with a positive diagonal. Note that no permutation matrix P is needed. Why?