Contents

Introduction

Beginning of Jan. 15, 2021

Big picture of MATH 501: *use computers to solve math problems*.

(1) Linear equations, differentiation, integration, differential equations, etc.

(2) **Approximation** & **optimization**.

Continuous vs. **Discrete**:

	Continuous (real world)	Discrete (computers)
Examples	Numbers in $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, etc. (Never-ending decimals)	Integers; floating-point numbers
Differentiation	$f'(x) := \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$	Difference quotient: compute the fraction with a given Δx; the smaller Δx is the more accurate the *approximation* is.
Continuous functions	$\sin x, e^x$, etc.	Discrete function approximations (pairs of $\{x, f(x)\}$ evaluated at specific values on the domain)

Chapter 2

Computer Arithmetic

Beginning of Jan. 20, 2021

2.1 Floating-Point Numbers & Roundoff Errors

Decimal system vs. **Binary system**:

$$427.325 = 4 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0 + 3 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$$

$$(1001.11101)_2 = 1 \cdot 2^3 + 1 \cdot 2^0 + 1^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-5}$$

Definition 2.1.1

In **Normalized Scientific Notation**, we denote x, a nonzero number, as $x := \pm r \cdot 10^n$ in decimal system where $1/10 \leqslant r < 1$, or $x := \pm q \cdot 2^m$ in binary system, where $1/2 \leqslant q < 1$. We call r "**mantissa**" and m the **exponent**. In a binary computer, both mantissa and exponent will be represented as binary numbers.

In our hypothetical computer `MARC-32` with 32 bits (1 word = 4 bytes = 32 bits (binary digits)). To recover x from its normalized scientific notation:

Sign of mantissa	Sign of exponent	Exponent(E)	Normalized mantissa excluding first digit (F)
1 bit	1 bit	7 bits	23 bits

Notice that in bianry system, the first digit of the normalized mantissa is *always* 1:

$$x = (-1)^S \cdot q \cdot 2^m \text{ where } q = (\overline{0.1F})_2 \text{ and } m = (-1)^S E$$

(where $\overline{0.1F}$ denotes the binary number with 1 as its first digit after the decimal point).

Limitations: all numbers that can be inputted into the computer using this method are called **machine numbers**.

(1) restriction on E: he absolute value of exponent $|m| = E$ can be at most $\sum_{n=0}^{6} 2^n = 2^7 - 1 = 127$. Since

$$10^{-38} \approx 2^{-127} \leqslant 2^m \leqslant 2^{127} \approx 10^{38}$$

and $q \in [1/2, 1)$, the range of all machine numbers is about $10^{-38} \leqslant |x| \leqslant 10^{38}$.

(2) restriction on q: since the first digit after decimal point is always 1 and we can store an additional 23 bits, we can keep 24 significant bits in the mantissa q. Because $2^{-24} \approx 10^{-7}$ we can roughly keep *seven decimal places*.

Therefore, all machine numbers are of form

$$x = \pm r \cdot 10^n \text{ where } \begin{cases} -38 \leqslant n \leqslant 38, \text{ and} \\ r = 0.r_1r_2r_3r_4r_5r_6r_7. \end{cases}$$

Integers: one bit for sign and 31 bits to record the actual integer: from 2^0 to 2^{30}. Hence the range is from $-(2^{31}-1)$ to $2^{31}-1 = 214783647$ in `MARC-32`.

Single-precision uses 1 word to store a number of 32 bits, whereas **double-prevision** uses 2 words to store words of a total of 64 bits.

Nearby Machine Numbers

Since the machine numbers are finite discrete numbers, if the input is some complicated x, the computer will need to choose a **nearby machine number** of x as its substitute. Cf. rounding.

$$x := (0.a_1a_2\ldots a_{25}a_{26})_2 \cdot 2^m \begin{cases} \overset{\text{chopping}}{\Longrightarrow} x' = (0.a_1a_2\ldots a_{24})_2 \cdot 2^m \\ \overset{\text{rounding}}{\Longrightarrow} x'' = \left((0.a_1a_2\ldots a_{24})_2 + 2^{-24}\right) \cdot 2^m \end{cases}$$

Between x' and x'', the computer will choose whichever is closer to x and define it as $x^* := \text{fl}(x)$. Note that $x'' - x' = 2^{-24} \cdot 2^m$. An example of rounding with 7 digits:

$$(0.\underbrace{0111101}_{\text{7digits}}1\ldots)_2 \overset{\text{rounding}}{\longrightarrow} (0.0111101)_2 + 2^{-7} = (0.0111110)_2.$$

Errors

For $x := q \cdot 2^m$, we have two types of errors:

(1) the **round-off error** is defined as

$$|x - x^*| \leqslant \frac{1}{2}|x'' - x'| = \frac{1}{2} \cdot 2^{-24} \cdot 2^m = 2^{m-25}$$

(2) the **relative error**/unit round-off error is defined as

$$\left|\frac{x - x^*}{x}\right| \leqslant \frac{2^{m-25}}{q \cdot 2^m} = \frac{2^{-25}}{q} \leqslant 2^{-24} \text{ since } q \geqslant \frac{1}{2}.$$

Beginning of Jan. 22, 2021

It follows that $x^* = \text{fl}(x) = x(1+\delta)$, where $|\delta| \leqslant \epsilon = 2^{-24}$. We define this ϵ to be the **unit round-off error**.

Overflow and Underflow

Suppose $x = \pm q \cdot 2^m$.

(1) If the exponent m is too large we say an **overflow** occurs. Computer will usually return `NaN`, not a number.

(2) If m is too small, (e.g., < -127), an **underflow** occurs. 0 will be returned.

Remark. Note that:

(1) Round-off errors must be expected to be present whenever data are read into a computer! Even simple numbers like $1/10$ cannot be stored exactly in `MARC-32` since

$$\frac{1}{10} = (0.0001\ 1001\ 1001 \dots)_2.$$

(2) Errors will accumulate as arithmetic operations are carried out. *We will talk about it later*.

(3) Distribution of machine numbers is uneven: denser near 0 and sparser when further away (log scale).

Algorithm to Test Machine Epsilon (HW1)

```
s = 1.0; //starting with exponent 0 (i.e. 2^0)
  for k = 1:100
    s = s/2; //keep decreasing the exponent by 1
    t = s + 1.0;
    if t <= 1.0 //detect the first time when 1.0+eps=1.0
      s = s*2;
      k = k-1; //-1 to get exponent of eps
      break
    end
  end
```

Floating-Point Arithmetic

Now we know $\text{fl}(x) = (1 + \delta x)$ for some $|\delta| \leqslant \epsilon$. The next natural question to think about is, *what about* $\text{fl}(x \otimes y)$? (Assuming $\otimes$ is one of the basic arithmetic operations.)

Example 2.1.2. Operation on two machine numbers:

$$\begin{aligned}
\text{fl}(x \otimes y) &= \text{fl}[\text{fl}(x) \otimes \text{fl}(y)] \\
&= (1 + \delta 3)[x(1 + \delta_1)y(1 + \delta_2)] \\
&= xy(1 + \delta_1)(1 + \delta_2)(1 + \delta_3) \\
&\sim (1 + \underbrace{\delta_1 + \delta_2 + \delta_3}_{|\cdot| \leqslant 3\epsilon})xy. \qquad \text{(since the cross-terms can be neglected)}
\end{aligned}$$

Theorem 2.1.3

Let $\{x_k\}_{k=0}^n$ be a set of n positive machine numbers, all of whose unit round-off error is ϵ. Then the relative round-off error of the sum $\sum_{i=1}^n x_i$ (in the usual way, no chopping or rounding in the process) is approximately $(1+\epsilon)^n - 1 \approx n\epsilon$ (again, we neglect terms with higher-order ϵ's.)

Beginning of Jan. 25, 2021

Proof. Let $S_k := x_0 + \cdots + x_k$ be the partial sum and let S_k^* be what the computer calculates for the k^{th} partial sum (not directly taking $\text{fl}(S_k)$). It follows that

$$\begin{cases} S_0 = x_0 \\ S_{k+1} = S_k + x_{k+1} \end{cases} \quad \text{and} \quad \begin{cases} S_0^* = x_0 & \text{(already a machine number)} \\ S_{k+1}^* = \text{fl}(S_k^* + x_{k+1}) & (S_k^* \text{ and } x_{k+1} \text{ also machine numbers}) \end{cases}$$

Notice that, although $S_k := x_0 + \ldots$ looks obvious to us and the other equations above look cumbersome to us, the opposite is true for computers, since the equation pairs above are defined recursively, something the computer is very good at.

Now we define

$$\rho_k := \frac{S_k^* - S_k}{S_k} \text{ and } \delta_k = \frac{S_{k+1}^* - (S_k^* + x_{k+1})}{S_k^* + x_{k+1}}.$$

Then $|\rho_k|$ is the relative error when approximating S_k by S_k^*, and $|\delta_k|$ is the relative error in approximating $S_k^* + x_{k+1}$ by $\text{fl}(\cdot)$. Then

$$\begin{aligned} \rho_{k+1} &= \frac{S_{k+1}^* - S_{k+1}}{S_{k+1}} \\ &= \frac{(S_k^* + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\ &= \frac{[S_k(1+\rho_k) + x_{k+1}](1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\ &= \frac{(S_k + x_{k+1})\delta_k}{S_{k+1}} + \frac{S_k\rho_k(1+\delta_k)}{S_{k+1}} \\ &= \delta_k + \rho_k\left(\frac{S_k}{S_{k+1}}\right)(1+\rho_k) \\ &\leqslant \delta_k + \rho_k(1+\delta_k) \end{aligned}$$

and since $|\delta_k| \leqslant \epsilon$ we have

$$|\rho_{k+1}| \leqslant \epsilon + |\rho_k|(1+\epsilon).$$

Therefore $|\rho_0| = 0$, $|\rho_1| \leqslant \epsilon$, $|\rho_2| \leqslant \epsilon + (1+\epsilon)\epsilon$, and inductively we see $|\rho_n| \leqslant \epsilon + (1+\epsilon) + \cdots + (1+\epsilon)^{n-1}$. Hence

$$|\rho_n| \leqslant \epsilon \sum_{k=0}^{n-1} (1+\epsilon)^k = \epsilon\left(\frac{\delta^n - 1}{\delta - 1}\right) = (1+\epsilon)^n - 1 = n\epsilon + \mathcal{O}(\epsilon^2).$$

Hence we are done. □

Suppose a number x is defined by infinite series where each term is real. For example

$$x := \frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}.$$

How do we approximate x? How many digits will be correct for sure?

(1) First stage: we make sure $|x - x_n| < 10^{-m}/2$ where x_n is the partial sum of the first n terms. (Complete math!)

(2) Second stage: we make sure $|x_n - x_n^*| < 10^{-m}/2$. (Numerical analysis)

2.2 Absolute & Relative Errors; Loss of Significance

Just like in last chapter, for some real number x, if it is approximated by another number (not necessarily a machine number) x^*, then the **error** is $x - x^*$. Then **absolute error** is $|x - x^*|$ and the **relative error** is $|(x - x^*)/x|$.

Loss of Significance

Suppose we are doing subtraction between two numbers that are close to each other. For example,

$$\begin{cases} x = .3721478693 \\ y = .3720230572 \end{cases} \implies x - y = .0001248121.$$

Now what if we input these numbers into a computer with decimal system and a 5-digit mantissa?

$$\begin{cases} \text{fl}(x) = .37215 \\ \text{fl}(y) = .37202 \end{cases} \implies \text{fl}(\text{fl}(x) - \text{fl}(y)) = .00013.$$

In this case the relative error is pretty large:

$$\left| \frac{x - y - \text{fl}(\text{fl}(x) - \text{fl}(y))}{x - y} \right| \approx 4\%.$$

Notice that originally when we did $x - y$, we calculated all the way till the last digit. However, when we calculated $\text{fl}(x) - \text{fl}(y)$, all digits starting from the 5^{th} become 0. That's where the huge error comes from.
On the other hand, since x and y are very close, so are $\text{fl}(x)$ and $\text{fl}(y)$, the first few digits are all 0. When we normalize $\text{fl}(x) - \text{fl}(y)$, $.00013$ becomes $.13000 \cdot 10^{-3}$, with 3 extra 0's supplemented on the right where they should have been something else.

To avoid these errors, we should avoid situations where accuracy can be ruined by a *subtraction between two nearly equal quantities*.

Example 2.2.1. As $x \to 0$, $f(x) := \sqrt{x^2+1} - 1$ also tends to 0. We therefore re-write the function by

$$f(x) := (\sqrt{x^2+1} - 1)\left(\frac{\sqrt{x^2+1}+1}{\sqrt{x^2+1}+1}\right) = \frac{x^2}{\sqrt{x^2+1}+1}.$$

Then we have solved the issue by re-writing $f(x)$ as a *fraction* instead of *subtraction*.
And as $x \to 0$, we can re-write $\sin(x+1) - \sin(1)$ by its Taylor series.

Beginning of Jan. 27, 2021

2.3 Stable and Unstable Computations; Conditioning

Numerical Instability

A numerical process is **unstable** if small errors made at one stage of the process are magnified in subsequence stages and seriously degrade the accuracy of the overall calculation.

Example 2.3.1. Consider the sequence of sequence of real numbers defined inductively by

$$\begin{cases} x_0 = 1, \quad x_1 = 1/3 \\ x_{n+1} = \dfrac{13}{3}x_n - \dfrac{4}{3}x_{n-1}. \end{cases}$$

Clearly, to us, the sequence is simply $x_n := (1/3)^n$. However, `MARC-32` will compute some ridiculous as it gives $x_{15} = 3.657493$, with a relative error of 10^8 (where the correct answer should be 3^{-15}).
At first, suppose x_0 and x_1 have small round-off error: $x_0 = 1 + \epsilon_0$ and $x_1 = 1/3 + \epsilon_1$. Then,

$$x_2 = \frac{13}{3}\left(\frac{1}{3} + \epsilon_1\right) - \frac{4}{3}(1 + \epsilon_0) = \frac{1}{9} + \frac{13}{3}\epsilon_1 - \frac{4}{3}\epsilon_2.$$

The term $13/3$ will grow *exponentially* as the sequence is computed recursively!

Using linear algebra and eigenvalues to solve $x_{n+1} = \dfrac{13}{3}x_n - \dfrac{4}{3}x_{n-1}$, we have the general solution

$$x_n = A(1/3)^n + B \cdot 4^n.$$

Where A, B are determined by x_0, x_1. If $x_0 = 1$ and $x_1 = 1/3$, we need to solve

$$\begin{cases} 1 = A + B \\ 1/3 = 1/3A + 4B \end{cases} \implies \begin{cases} A = 1 \\ B = 0. \end{cases}$$

Now suppose $x_0 = 1 + \epsilon_1$ and $x_1 = 1/3 + \epsilon_2$. Then

$$\begin{cases} 1 + \epsilon_1 = A + B \\ 1/3 + \epsilon_2 = 1/3A + 4B \end{cases} \implies \begin{cases} A = 1 + \delta_1 \\ B = \delta_2. \end{cases}$$

It follows that the general solution $x_n = A(1/3)^n + B \cdot 4^n$ gives

$$x_n^* = (1 + \delta_1)(1/3)^n + \delta_2 \cdot 4^n,$$

where the second term again blows up.
However, sometimes we are able to produce a stable process:

Example 2.3.2. Now let $x_0 = 1$ and $x_1 = 4$. Then $x_n = A(1/3)^n + B \cdot 4^n$ where $A = 0$ and $B = 1$. Then $x_n = 4^n$, and $x_n^* = \delta_1(1/3)^n + (1 + \delta_2)4^n$. Then, $x_n^* - x_n = \delta_1(1/3)^n + \delta_2 \cdot 4^n$, and

$$\left|\frac{x_n^* - x_n}{x_n}\right| \leqslant \frac{\delta_1(1/3)^n}{4^n} + \delta_2 \approx \delta_2.$$

In other words, if the answer x_n's are large enough, they are over to overwhelm the errors — which are also

large, but not on the same magnitude of the x_n's.

Conditioning

Used informally to indicate how sensitive the solution of a problem may be to small relative changes in the input. A problem is **ill conditioned** if small changes in data can produce large changes in the answers. It's a "*quantitative description of how unstable a problem is*".

Example 2.3.3. The problem asks us to evaluate a function f at a point x. Question for conditioning: if x is perturbed slightly, say by $+h$, what is the effect on $f(x)$?
We consider the difference $f(x+h)-f(x)$ which, by MVT, equals $f'(\xi)h \approx hf'(x)$ for some $\xi \in [x, x+h]$, assuming $f'(\xi)$ is not too large so that $f'(\xi) \approx f'(x)$. Then the relative size of the pertubation is

$$\frac{f(x+h)-f(x)}{f(x)} \approx \frac{hf'(x)}{f(x)} = \underbrace{\left[\frac{xf'(x)}{f(x)}\right]}_{\text{conditional number}} \overbrace{\frac{h}{x}}^{\text{relative size of pertubation}}.$$

Example 2.3.4. Another type of condition number is associated with solving linear system $Ax = b$, assuming A is invertible. (x is output and b is input.) Suppose a perturbation gives $b+h$. Then

$$A\tilde{x} = b+h \implies \tilde{x} = A^{-1}(b+h).$$

Let $\|\cdot\|$ be any norm. Then

$$\|\tilde{x}-x\| = \|A^{-1}(b+h) - A^{-1}(b)\| = \|A^{-1}h\|.$$

Let $\|A^{-1}\|$ denote the *matrix norm* of A^{-1} (which we'll discuss later). Then (by Cauchy-Schwarz)

$$\begin{aligned}
\frac{\|\tilde{x}-x\|}{\|x\|} = \frac{\|A^{-1}h\|}{\|x\|} &\leqslant \frac{\|A^{-1}\|\|h\|\|b\|}{\|x\|\|b\|} \\
&= \frac{\|A^{-1}\|\|Ax\|\|h\|}{\|x\|\|b\|} \\
&\leqslant \frac{\|A^{-1}\|\|A\|\|x\|}{\|x\|} \cdot \frac{\|h\|}{\|b\|} \\
&= \underbrace{\|A^{-1}\|\|A\|}_{\text{conditional number}} \cdot \overbrace{\frac{\|h\|}{\|b\|}}^{\text{relative size of pertubation}}.
\end{aligned}$$

We denote the conditional number of a matrix A as $\kappa(A) := \|A\|\|A^{-1}\|$, as shown above.

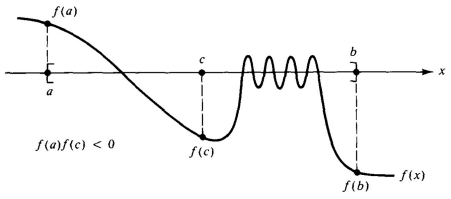
Chapter 3

Solution of Nonlinear Equations

Beginning of Jan. 29, 2021

3.1 Bisection Method

If f is continuous on $[a, b]$ and if $f(a)f(b) < 0$, then there exists $\xi \in [a, b]$ such that $f(\xi) = 0$.
Goal: find $\xi \in (a, b)$ such that $f(\xi) = 0$, assuming $f(a)f(b) < 0$, i.e., both nonzero with different signs.
WLOG assume $f(a) > 0$ and $f(b) < 0$. We define $c := (a + b)/2$. If $f(c) = 0$ we are done. If not, either $f(c) > 0$ or $f(c) < 0$. Then $f(c)$ has a different sign with either $f(a)$ or $f(b)$, and we are able to inductively conduct this bisection method again.



However, notice that, although we can find *one* root, we cannot tell *which* root it is, and we may not find *all* the roots. See the figure above; we can inductively find the root in $[a, c]$ but not those roots in $[c, b]$.

input $a, b, M, \delta, \varepsilon$
$u \leftarrow f(a)$
$v \leftarrow f(b)$
$e \leftarrow b - a$
output a, b, u, v
if sign $(u) =$ sign (v) **then stop**
for $k = 1, 2, \ldots, M$ **do**
  $e \leftarrow e/2$
  $c \leftarrow a + e$
  $w \leftarrow f(c)$
  output k, c, w, e
  if $|e| < \delta$ **or** $|w| < \varepsilon$ **then stop**
  if sign $(w) \neq$ sign (u) **then**
    $b \leftarrow c$
    $v \leftarrow w$
  else
    $a \leftarrow c$
    $u \leftarrow w$
  end if
end

On the left is a pseudocode for the bisection algorithm. Things to notice:

(1) When computing midpoint, we use $c := a + (b - a)/2$ as opposed to $c := (a + b)/2$ to prevent potential overflow when both a and b are large.

(2) When comparing the sign, we simply compare the sign as opposed to testing whether $f(a)f(b) > 0$ because multiplication takes much more time.

(3) Three stopping criteria: number of iteration bounded by M to prevent the algorithm running forever, and the ϵ and δ bounds on $|e|$ and $|w|$ to control the accuracy of the algorithm. Each one can be independently changed in order to meet our specific demand.

Error Analysis

If define the interval after k^{th} iteration as $[a_k, b_k]$ we see that

$$a_0 \leqslant a_1 \cdots \leqslant b_0 \text{ and } b_0 \geqslant b_1 \geqslant \cdots \geqslant a_0.$$

In addition,

$$b_{n+1} - a_{n+1} = \frac{1}{2}(b_n - a_n). \tag{*}$$

Therefore, since $\{a_n\}$ is a bounded, increasing sequence and $\{b_n\}$ a bounded, decreasing sequence, $\lim_{n\to\infty} a_n$ and $\lim_{n\to\infty} b_n$ both exist. Applying (*) iteratively gives

$$\lim_{n\to\infty} b_n - \lim_{n\to\infty} a_n = \lim_{n\to\infty} 2^{-n}(b_0 - a_0) = 0.$$

We claim that $r = \lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$ is a solution. Indeed, by definition, $0 \geqslant f(a_n)f(b_n)$ for all n and so $0 \geqslant f(r)^2$.

Beginning of Feb. 1, 2021

Theorem 3.1.1

If $[a_0, b_0]$, $[a_1, b_1]$,... denote the intervals in the bisection method, then $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n$ (i.e., exist and equal) and the limits represent a root r of f. If c_n is the midpoint of $[a_n, b_n]$ then

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

Example 3.1.2. Suppose the bisection method starts with $[50, 63]$. How many steps are necessary to compute a root with relative accuracy in 10^{-12}?

Solution

We want to make sure $|r - c_n|/|r| \leqslant 10^{-12}$. Since $r \geqslant 50$, it suffices thos how $|r - c_n|/50 \leqslant 10^{-12}$. By the theorem

$$|r - c_n| \leqslant 2^{-(n+1)}(b_0 - a_0) \implies 2^{-(n+1)}\frac{13}{50} \leqslant 10^{-12} \implies n \geqslant 37.$$

3.2 Newton's Method

Goal: Simple. Solve $f(x) = 0$.

Let $f \in C^2$ and assume $f(r) = 0$. Then for h close to r, by Taylor expansion

$$0 = f(r) \approx f(x + h) = f(x) + hf'(x) + \mathcal{O}(h^2).$$

Then

$$0 \approx f(x) + hf'(x) \implies h \approx -\frac{f(x)}{f'(x)}.$$

Since $r = x + h$, we see that $x - \dfrac{f(x)}{f'(x)}$ is a *better approximation* of r.

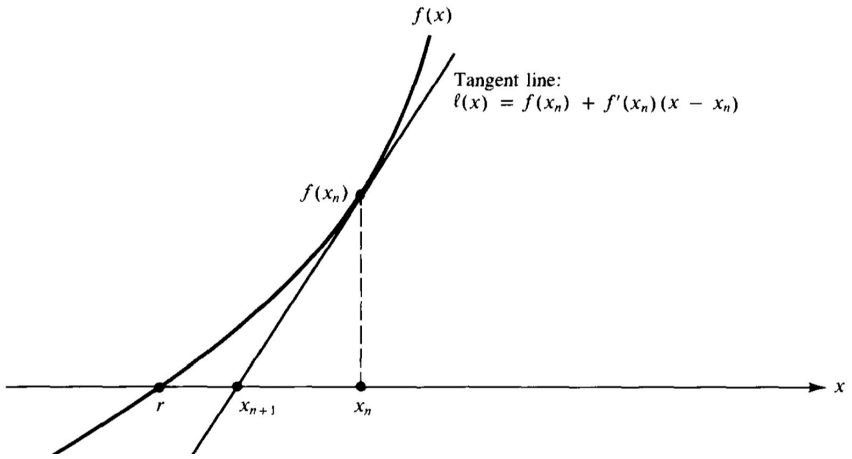
Definition 3.2.1

The **Newton's Method** is defined inductively by

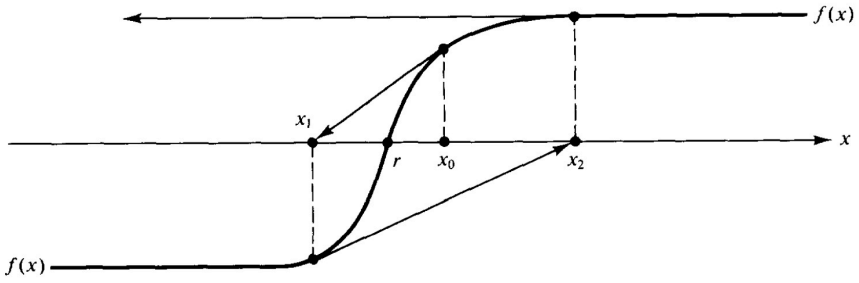
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

> **Remark.** Newton's Method depends on the initial value x_0. If it's far away from r, the Taylor expansion doesn't work and so won't Newton's method. We will discuss about this in detail later.

Newton's Method is very efficient: the example on p.65 shows that for $f(x) = e^x - 1.5 - \arctan(x)$, picking an initial guess with $f(x) \approx -0.7$ iterates to $f(x_6) \approx -1 \cdot 10^{-20}$ and $f(x_7)$ with error beyond machine epsilon.
A graphical representation of a sufficiently close x_0 where $\{x_n\}$ converges to r:



Again, recall that x_0 needs to be *sufficiently close* to r. An example where Newton's Method breaks down:



Error Analysis

Let $e_n = x_n - r$ be the error of x_n. Then

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

By Taylor's theorem we have

$$0 = f(r) = f(x_n - e_n) = f(x_n) - e_n f'(x_n) + \frac{1}{2} e_n^2 f''(\xi_n)$$

for some ξ_n between x_n and r. Therefore

$$e_{n+1} = \frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}e_n^2 \approx \frac{1}{2}\frac{f''(r)}{f'(r)}e_n^2 =: e_n^2.$$

We call this type of convergence, i.e., $e_{n+1} \sim e_n^2$ **quadratic convergence**. This converges *really fast*.

Theorem 3.2.2

Let $f \in C^2$ and r be a **simple zero** of f, i.e., $f'(r) \neq f(r) = 0$. Then there exists a neighborhood of r and a constant C such that if $x_0 \in (r - C, r + C)$ then the successive points become steadily closer to x and

$$|x_{n+1} - r| \leqslant C(x_n - r)^2.$$

Proof. Idea: we know $e_{n+1} = \frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}e_n^2$. If e_n is small and the same holds for the middle term, then $e_{n+1} < e_n$. Define

$$c(\delta) = \frac{1}{2}\frac{\max_{|x-r|\leqslant\delta}|f''(x)|}{\min_{|x-r|\leqslant\delta}|f'(x)|}$$

where $\delta > 0$. Notice that since $f \in C^2$ both the numerator and the denominator attain their extrema, and for small $\delta > 0$, since $f'(r) \neq 0$, this fraction is well-defined (and so finite). If necessary, further decrease $\delta > 0$ to ensure that $\delta c(\delta) < 1$.

If $|x_0| \leqslant \delta$ then $|e_0| \leqslant \delta$ and $|\xi_0 - r| \leqslant \delta$. Then,

$$\frac{|f''(\xi_0)/f'(x_0)|}{2} \leqslant c(\delta)$$

and

$$|x_1 - r| = |e_1| \leqslant e_0^2 c(\delta) \leqslant \delta c(\delta)|e_0| < |e_0| \leqslant \delta.$$

Doing this inductively, we have $|e_n| \leqslant (\delta c(\delta))^n |e_0|$. □

Beginning of Feb. 3, 2021

Theorem 3.2.3

If $f \in C^2$ is increasing, convex, and has a zero, then the zero is unique, and the Newton iteration works with any starting point.

Proof. By assumption, $f''(x) > 0$ for all x and $f'(x) > 0$. Therefore, since

$$e_{n+1} \approx \frac{1}{2}\frac{f''(r)}{f'(r)}e_n^2 > 0,$$

$x_n > r$ for all $n \geqslant 1$. Then since f is increasing, $f(x_n) > f(r) = 0$. Also recall that

$$e_{n+1} = \frac{e_n f'(x_n) - f(x_n)}{f'(x_n)} \implies e_{n+1} < e_n.$$

Therefore $\{e_n\}$ is a strictly decreasing and bounded from below, it converges to some e. Similarly $\{x_n\} \to x$. The above equation then suggests

$$e = e - \frac{f(x)}{f'(x)} \implies f(x) = 0.$$

□

Remark. That $f \in C^1$ is necessary for Newton's method since $x_{n+1} := x_n - \dfrac{f(x_n)}{f'(x_n)}$. Note that $f \in C^2$ is not a necessary condition, i.e., we won't run into problems if $f \in C^1 \setminus C^2$. The only reason we need $f \in C^2$ is because we want Newton's method to be fast (i.e., achieves quadratic convergence) since we do need the extra order of differentiability to apply Taylor's theorem to guarantee the existence of ξ_n between x_n and r for the term $e_n^2 f''(\xi_n)/2$.

Example 3.2.4. We can apply Newton's method to solve for square roots. To compute $\sqrt{R}$ we look for roots of $x^2 - R$. Then,

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

For example, for $\sqrt{17}$ we start with $x_0 = 4$. With 4 iterations we get x_4 with accuracy $< 10^{-26}$.

Implicit Functions

Newton's method can also be applied to solving implicit functions.
For a given x, we want to solve $G(x, y) = 0$. With a nice enough starting point y_0, we define

$$y_{n+1} := y_n - \frac{G(x, y_n)}{\partial G(x, y_n)/\partial y}.$$

Compare this with the formula $x_{n+1} := x_n - f(x_n)/f'(x_n)$.

Systems of Nonlinear Equations

Suppose we have $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$ simultaneously. This is equivalent to solving

$$\mathcal{F}(X) = \mathcal{F}(x_1, x_2) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Similar to before, once we have an initial guess X_1, we can apply Newton's method to get better approximations by

$$X_{n+1} = X_n - \frac{\mathcal{F}(X)}{\mathcal{F}'(X)}$$

where dividing by the so-called $\mathcal{F}'(X)$ is defined by multiplying (on the left) by the inverse of Jacobian.

3.3 Secant Method

Recall that in Newton's method we defined $x_n + 1 := x_n - f(x_n)/f'(x_n)$. It is completely possible that $f'(x_n) = 0$, in which case we run into a problem. Some of the workarounds:

(1) Simply replace $f'(x_n)$ by approximating x_n with $x_n + h$:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{h}{f(x_n + h) - f(x_n)} \text{ for some small } h.$$

Here we need to keep picking smaller h since as $e_n \to 0$, the relative error caused by h increases.

(2) Steffensen's iteration:

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

which also achieves quadratic convergence without defending on $f''(x)$. This is also similar to above in the sense that

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

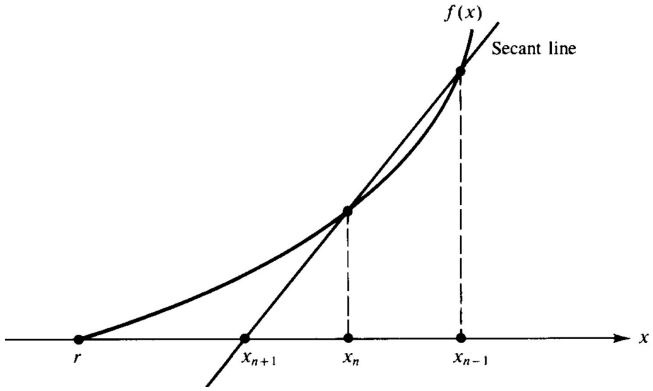
Unlike above, we do not need to worry about picking the appropriate h for each x_n since when $x_n \to r$, $f(x_n) \to 0$ itself.

Beginning of Feb. 5, 2021

Similar to Steffensen's iteration, as $n \to \infty$, $f(x_n) - f(x_{n-1})$ also tends to 0. Then we have the **secant method**

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

where the $[\cdot]$ denotes the slope of the secant line drawn between $(x_n, f(x_n))$ and $(x_{n-1}, f(x_{n-1}))$.



Intuitively from the diagram above, the secant method is not as fast as Newton's method.

Error Analysis

Assuming r is a simple zero, i.e., $f'(r) \neq f(r) = 0$, and $f \in C^2$. Rewrite x_{n+1} as

$$x_{n+1} = x_n - f(x_n)\left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right] = \frac{f(x_n)x_{n-1} - f(x_{n-1})f_n}{f(x_n) - f(x_{n-1})}.$$

Then,

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

Factoring out $e_n e_{n-1}$ and multiplying $(x_n - x_{n-1})/(x_n - x_{n-1})$ gives

$$e_{n+1} = \left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right]\left[\frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}}\right] e_n e_{n-1}.$$

The blue term can be roughly thought as $1/f'(r)$. For the second one, notice that

$$f(x_n) = f(r + e_n) = f(r) + e_n f'(r) + \frac{e_n^2 f''(r)}{2} + \mathcal{O}(e_n^3).$$

Substituting $f(r) = 0$ into the equation,

$$\frac{f(x_n)}{e_n} = f'(r) + \frac{e_n f''(r)}{2} + \mathcal{O}(e_n^2).$$

Similarly,

$$\frac{f(x_{n-1})}{e_{n-1}} = f'(r) + \frac{e_{n-1} f''(r)}{2} + \mathcal{E}_{n-1}^2.$$

Therefore their difference is approximately $(e_n - e_{n-1})f''(r)/2$ and

$$\frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}} \approx \frac{e_n - e_{n-1}}{x_n - x_{n-1}} \frac{f''(r)}{2} = \frac{f''(r)}{2}.$$

Therefore

$$e_{n+1} \approx \frac{1}{2}\frac{f''(r)}{f'(r)} e_n e_{n-1} =: C e_n e_{n-1}.$$

Now we determine the order of convergence:

$$|e_n|^\alpha \sim |e_{n+1}| \sim |C e_n e_{n-1}| \sim C|e_n||e_n|^{1/\alpha}$$

and so $\alpha = 1 + 1/\alpha$, of which the solution is $\alpha = (\sqrt{5} + 1)/2$. Since this number $= 1.618 < 2$, we see that indeed the secant method is not as strong as Newton's method (quadratic with exponent 2), but it's still better than the bisection method (linear with $e_{n+1} \approx e_n/2$).

However, unlike Newton's method where in each step the computer needs to evaluate both $f(x_n)$ and $f'(x_n)$, in the secant method the computer only needs to compute $f(x_n)$ since $f(x_{n-1})$ has been already evaluated. (*The basic arithmetic computations are far lighter burden than these computations*.) If we compensate this by comparing one step of Newton's method with two steps of the secant method, we see

$$|e_{n+1}| \sim |e_{n+1}|^\alpha \sim |e_n|^{\alpha^2}$$

in which $\alpha^2 \approx 2.618 > 2$, and the this is much faster than Newton's quadratic convergence.

> **Remark.** The above holds when we don't have **parallel computing**, i.e., only using one kernel. However, if parallel computing is allowed, Newton's method is once again faster — $f(x_n)$ and $f'(x_n)$ can be simultaneously evaluated, whereas $f(x_n)$ and $f(x_{n+1})$ cannot be done by two kernels at the same time. The secant method does not benefit from parallel computing.
>
> Therefore, if we want to design an algorithm, it would be better if our algorithm becomes more efficient as more kernels are involved.

Beginning of Feb. 8, 2021

3.4 Fixed Points and Functional Iteration

Do we have a framework to design our own iterative algorithm? How do we find other iteration methods to solve nonlinear equations?

Definition 3.4.1

An algorithm defined by an equation of form $x_{n+1} = F(x_n)$ is called **functional iteration**. For example, Newton's method and Steffen's method are both functional iterations:

$$F(x) = x - \frac{f(x)}{f'(x)} \text{ and } F(x) = x - \frac{(f(x))^2}{f(x + f(x)) - f(x)}$$

whereas the secant method cannot: it requires two inputs, namely x_n and x_{n-1}.

For the secant method, we instead consider a vector function:

$$y_{n+1} := [x_{n+1}, x_n] = F(y_n) = [F_1(y_n), F_2(y_n)]$$

by

$$F_1(y_n) := x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \text{ and } F_2(y_n) = x_n.$$

Definition 3.4.2

We say s is a **fixed point** of F if $F(s) = s$.

It follows that if F is continuous and $\{x_n\} \to s$, then

$$F(s) = F(\lim_{n\to\infty} x_n) = \lim_{n\to\infty} F(x_n) = \lim_{n\to\infty} x_{n+1} = s.$$

The main idea is to design some F such that the fixed point of F is the root of the equation $f(x) = 0$. For example consider $F(x) = x - f(x)$ where $F(r) = r - f(r) = r$ (it may not work, but the idea is to look for F's like this).

One sufficient condition for convergence of $\{x_n\}$ (there are more):

Definition 3.4.3

A mapping F is said to be **contractive** if there exists $\lambda < 1$ such that

$$|F(x) - F(y)| \leqslant \lambda |x - y|$$

for all x, y in the domain of F.

Theorem 3.4.4: Contractive Mapping Theorem

Let F be a contractive mapping of a closed set $C \subset \mathbb{R}$ to C. Then F has a fixed point. Furthermore, this fixed point is the limit of every sequence obtained from $x_{n+1} = F(x_n)$ with $x_0 \in C$.

Proof. Since $|x_n - x_{n-1}| \leqslant \lambda |x_{n-1} - |x_{n-2}||$, repeating this argument gives

$$|x_n - x_{n-1}| \leqslant \lambda^{n-1} |x_1 - x_0|.$$

Therefore,

$$|x_m - x_n| = \left| \sum_{i=n+1}^{m} x_i - x_{i-1} \right| \leqslant \frac{\lambda^n}{1-\lambda} |x_1 - x_0|.$$

From this we see that $\{x_n\}$ is Cauchy and is therefore convergent in $\mathbb{R}$. Since C is closed, the limit, say s, lies within C. By what we've derived previously. $F(s) = s$, i.e., s is a fixed point of F. □

Beginning of Feb. 10, 2021

Example 3.4.5. Use the contractive mapping theorem to compute a fixed point of

$$F(x) = 4 + \frac{\sin 2x}{3}.$$

Solution

First we verify that F is contractive. For $x, y \in \mathbb{R}$,

$$|F(x) - F(y)| = \frac{|\sin 2x - \sin 2y|}{3} = \frac{2|\cos 2\xi|}{3} \cdot |x - y| \leqslant \frac{2}{3}|x - y|$$

for some ξ between x and y, so indeed this mapping is contractive. Alternatively, sum-to-product gives

$$|\sin 2x - \sin 2y| = |\sin(x+y)\cos(x-y)| \leqslant |\sin(x+y)||\cos(x-y)|.$$

Therefore any $\{x_n\}$ converges to some $x \in \mathbb{R}$. Then the rest can be done by computer.

Example 3.4.6. Let $p > 1$. Evaluate

$$x = \cfrac{1}{p + \cfrac{1}{p + \cfrac{1}{p + \dots}}}$$

Solution

Define $x_1 := 1/p$, $x_2 = 1/(p + 1/p)$, and so on. It follows that $x_n \to x$. If we can find a contractive mapping then the problem is solved. Define

$$F(x_n) = x_{n+1} := \frac{1}{p + x_n}.$$

Notice that $F : [0, 1/p] \to [0, 1/p]$. It is contractive, since

$$|F(x) - F(y)| = \left| \frac{1}{p+x} - \frac{1}{p+y} \right| = \left| \frac{y-x}{(p+x)(p+y)} \right| \leqslant \frac{|y-x|}{p^2}$$

where $1/p^2$ is a constant < 1. The fixed point is simply s where $s = F(s) = 1/(p+s)$ so $s^2 + ps - 1 = 0$, i.e.,

$$s = \frac{-p \pm \sqrt{p^2+4}}{2} \implies s = \frac{-p + \sqrt{p^2+4}}{2}, \text{ assuming } s > 0.$$

Error Analysis

Suppose F has a fixed point s and a sequence $\{x_n\}$ is a sequence defined by $x_{n+1} = F(x_n)$. Let s be the fixed point and let $e_n := x_n - s$.

If F' exists and is continuous, then by MVT

$$x_{n+1} - s = F(x_n) - F(s) = F'(\xi_n)(x_n - s) = F'(\xi_n)e_n$$

for some $\xi_n \in [s, x_n]$.

Therefore if $|F'(x)| < 1$ for all x, $e_n \to 0$.

Furthermore, if e_n is small then $\xi_n \approx s$ and $F'(x_n) \approx F'(s)$. From above we see that the convergence would be *really* quick if $F'(s)$ is small. Ideal situation: $F'(s) = 0$. Let q be such that $F^{(q)}(s) \neq 0$ but $F^{(k)}(s) = 0$ for all $k < q$. Taylor series for $F(x_n)$ expanded about s gives

$$\begin{aligned} e_{n+1} &= F(x_n) - F(s) \\ &= F(s + e_n) - F(s) \\ &= \left[F(s) + e_n F'(s) + e_n^2 F''(s)/2 + \dots\right] - F(s) \\ &= e_n F'(s) + \frac{e_n^2}{2} F''(s) + \cdots + \frac{e_n^{q-1}}{(q-1)!} F^{(q-1)}(s) + \frac{e_n^q}{q!} F^{(q)}(\xi_n). \end{aligned}$$

Therefore if the derivatives of $F(s)$ vanishes up to $(q-1)^{\text{th}}$ order, this algorithm is *at least* $q+1^{\text{th}}$ order convergent. For example, consider Newton's method again: $x_{n+1} = F(x_n) = x_n - f(x_n)/f'(x_n)$. We know this has quadratic convergence. We have assumed that $F'(s) = 0$ so this is *at least* quadratically convergent. Let s be a fixed point of F so that $F(s) = 0$. Then

$$F'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{f(x)^2} = \frac{f(x)f''(x)}{f'(x)^2} \implies F'(s) = 0$$

and so by above, Newton's method is at least quadratically convergent.

Chapter 4

Solving Systems of Linear Equations

4.1 Matrix Algebra

Beginning of Feb. 12, 2021

Recall from linear algebra that necessary conditions for the uniqueness of solution to $Ax = b$ include that A has at least as many rows as columns, i.e., $A_{m\times n}$ with $m \geqslant n$.

Some quick recap:

Definition 4.1.1

The **elementary operations** for a linear system include the following:

(1) Interchanging two equations in the system: $\mathcal{E}_i \leftrightarrow \mathcal{E}_j$,

(2) Multiplying an equation by a nonzero number: $\lambda\mathcal{E}_i \to \mathcal{E}_i$, and

(3) Adding one equation to the multiple of another: $\mathcal{E}_i + \lambda\mathcal{E}_j \to \mathcal{E}_i$.

Theorem 4.1.2

If one system of equations is obtained by a finite sequence of elementary operations (for an infinite counterexample: consider infinite iteration with $\lambda = 1/2$), then the two systems are equivalent, i.e., having the same solution.

Definition 4.1.3

The **elementary row operations** for a matrix includes three operations analogous to above: interchanging rows, multiplying one row by λ, and adding one row to λ times another. They correspond to multiplying by

the **elementary matrix** on the left. Take 3×3 matrices for example:

$$\mathcal{E}_2 \leftrightarrow \mathcal{E}_3 : \begin{bmatrix} 1 & & \\ & & 1 \\ & 1 & \end{bmatrix}; \quad \lambda\mathcal{E}_2 \to \mathcal{E}_2 : \begin{bmatrix} 1 & & \\ & \lambda & \\ & & 1 \end{bmatrix}; \quad \mathcal{E}_3 + \lambda\mathcal{E}_2 \to \mathcal{E}_3 : \begin{bmatrix} 1 & & \\ & 1 & \\ & \lambda & 1 \end{bmatrix}$$

where blank entries correspond to 0.

Definition 4.1.4

For $A_{m\times n}$ and $B_{n\times m}$, if $AB = I_{m\times m}$ we say B is the **right** inverse of A and A the **left inverse** of B.

Remark. A short, wide matrix usually does not have a unique right inverse, and a long, thin matrix usually does not have a unique left inverse:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \alpha & \beta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

Theorem 4.1.5

A square matrix has at most one right inverse.

Proof. Let $AB = I$ where A, B, I are $n \times n$. Let $A^{(j)}$ be the j^{th} column of A and $I^{(k)}$ the k^{th} column of I. Since $AB = I$, this gives

$$\sum_{j=1}^{n} b_{jk} A^{(i)} = I^{(k)}, 1 \leqslant k \leqslant n.$$

Therefore each column of I is a linear combination of columns of A. Since $C(I) = \mathbb{R}^n$, we also have $C(A) = \mathbb{R}^n$. On the other hand, $\dim C(A) \leqslant n$. It follows that columns of A must be linearly independent, and the equation above is uniquely determined by a bunch of b_{jk}'s, i.e., the right inverse must be unique. □

Theorem 4.1.6

If A, B are square matrices with $AB = I$ then $BA = I$, i.e., the right inverse of a square matrix, should it exist, is also the left inverse. Because of this, when talking about square matrices, we simply consider inverses without specifying right or left inverses.

Proof. Simple. Let $C = BA - I + B$. Then $AC = ABA - AI + AB = A - A + I = I \implies C = B$. □

Eliminations

If E_n's are elementary matrices and $E_m E_{m-1} \dots E_1 A = I$, then

$$A^{-1} = E_m E_{m-1} \dots E_1.$$

This can be implemented into Gauss-Jordan elimination:

$$\left[A \mid I\right] \to \left[I \mid A^{-1}\right] = \left[AA^{-1} \mid IA^{-1}\right].$$

Theorem 4.1.7

Let A be $n \times n$. TFAE:

(1) A^{-1} exists, i.e., A is nonsingular,

(2) $\det(A) \neq 0$ (determinant = product of eigenvalues; see last part)

(3) $C(A^T) = C(A) = \mathbb{R}^n$,

(4) $A : \mathbb{R}^n \to \mathbb{R}^n$ is injective, or surjective, or bijective,

(5) $Ax = 0 \implies x = 0$,

(6) $Ax = b$ has a unique solution, i.e., $N(A) = \{0\}$,

(7) A is a product of elementary matrices, and

(8) 0 is not an eigenvalue of A.

Partitioned Matrices

Multiplying partitioned matrices works just like multiplying normal matrices, as long as the submatrices "align nicely":

$$\begin{bmatrix} A_{m_1 \times n_1} & A_{m_2 \times n_1} \\ A_{m_1 \times n_2} & A_{m_2 \times n_2} \end{bmatrix} \begin{bmatrix} B_{n_1 \times k_1} & B_{n_2 \times k_1} \\ B_{n_1 \times k_2} & B_{n_2 \times k_2} \end{bmatrix} = \begin{bmatrix} C_{m_1 \times k_1} & C_{m_1 \times k_2} \\ C_{m_2 \times k_1} & C_{m_2 \times k_2} \end{bmatrix}.$$

Beginning of Feb. 17, 2021

4.2 The LU and Cholesky Factorizations

Consider $Ax = b$ where A is $n \times n$, a system of n linear equations in n unknowns $x_1, \ldots, x_n$.

Easy-to-solve Systems

(1) If A is a diagonal matrix then we immediately have the solution $x_n = b_n / a_{n,n}$.

(2) Lower triangular or upper triangular A, assuming the diagonal entries are nonzero: we start with the row with only one nonzero entry, solve the corresponding x_i (either x_1 or x_n), then do forward or backward substitution.

(3) Permuted lower/upper triangular A, again assuming A before permutation has nonzero diagonal entries: we simply reorder the permutation and apply (2).

LU Factorization

If we can decompose A into LU where L is lower triangular and U upper triangular, then solving $Ax = b$ is equivalent to solving $LUx = b$. We first solve $L(Ux) = b$ to get the solution $Ux = z$, which is easy since L is triangular, and then we solve $Ux = z$, which is again easy since U is triangular. It follows that

$$a_{ij} = \sum_{s=1}^{n} \ell_{is} u_{sj} = \sum_{s=1}^{\min(i,j)} \ell_{is} u_{sj} \text{ since } L \text{ and } U \text{ are triangular.}$$

Since A is $n \times n$ we have n^2 equations, and we have $n(n+1)$ unknowns from the nonzero entries of L and U. More unknowns than solutions, so the solutions are not unique. For convenience we set L to be *unit lower triangular* (diagonal entries 1), known as **Doolittle's factorization**, or set U to be *unit upper triangular*, known as **Crout's factorization**.

$$L = \begin{bmatrix} \ell_{11} & 0 & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & \ell_{nn} \end{bmatrix} \qquad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Suppose we adopt Doolittle's factorization and assume $\ell_{n,n} = 1$.

Staying with the first row of U: since the first row of L contains only one nonzero entry ℓ_{11}, $a_{11} = \ell_{11} + u_{11}$, we immediately get the value of u_{11}. Likewise, $a_{12} = \ell_{11} u_{12}$, and so on. From this we obtained the first row of U.

Now we stay with the first column of L (notice the parallel computing here! n^{th} row of U, along with n^{th} column of L). Since the first row of U contains only one nonzero entry u_{11}, we see $a_{21} = \ell_{21} u_{11}$, $a_{31} = \ell_{31} u_{11}$, and so on. Therefore we obtain the entire first column of L.

Now we repeat the process, computing the second row of U and second column of L, and then n^{th} row of U and n^{th} column of L.

Remark. We need A to be "nice" to have such LU decomposition. See theorem below.

Theorem 4.2.1

If all n learding *leading principal minors* (the truncated $m \times m$ top-left submatries of A) of the $n \times n$ matrix A are nonsingular, then A has a LU-decomposition.

Proof. We prove by induction. Let A_k be the k^{th} leading principal minor of A (the top-left $k \times k$ submatrix) and let L_k, U_k be the k^{th} leading principal minors of L and U.

Immediately we see $A_1 = L_1 U_1$ as long as A_1 is nonsingular.

Now suppose $A_k = L_k U_k$; it remains to show $A_{k+1} = L_{k+1} U_{k+1}$. By matrix block multiplication,

$$\begin{bmatrix} A_{k\times k} & a_{k\times 1} \\ b^T_{1\times k} & a_{k+1,k+1} \end{bmatrix} = \begin{bmatrix} L_{k\times k} & 0_{k\times 1} \\ \ell^T_{1\times k} & \ell_{k+1,k+1} \end{bmatrix} \begin{bmatrix} U_{k\times k} & u_{k\times 1} \\ 0_{1\times k} & u_{k+1,k+1} \end{bmatrix}.$$

Beginning of Feb. 19, 2021

Then,

(1) $A_{k\times k} = L_{k\times k}U_{k\times k}$,

(2) $a_{k\times 1} = L_{k\times k}u_{k\times 1}$,

(3) $b^T_{1\times k} = \ell^T_{1\times k}U_{k\times k}$, and

(4) $a_{k+1k+1} = \ell^T_{1\times k}u_{k\times 1} + \ell_{k+1,k+1}u_{k+1,k+1}$.

By assumption A_k is nonsingular, and since $A_k = U_kL_k$, so are these triangular matrices ($\det(A_k) = \det(L_k)\det(U_k)$). Hence (2) gives $u_{k\times 1} = (L_{k\times k})^{-1}a_{k\times 1}$, and (3) gives $\ell^T_{1\times k} = b^T_{1\times k}(U_{k\times k})^{-1}$. For (4), once we fix ab arbitrary choice of $\ell_{k+1,k+1}$, say 1, then we can solve for $u_{k_1,k+1}$. Then we have obtained a LU decomposition for A_{k+1}. □

Cholesky Factorization

Theorem 4.2.2

If A is a real, symmetric, and positive definite matrix, then it has a unique factorization $A = LL^T$ where L is lower triangular with a positive diagonal.

Proof. Recall that A is positive definite if $x^TAx > 0$ for all nonzero vector x. It follows that $x \neq 0 \implies Ax \neq 0$ and so A is invertible. Furthermore, by using vectors of form $x = (x_1, x_2, \dots, x_k, 0, \dots)^T$ we see that all leading principal minors A_k's are also positive definite and thus invertible. Indeed, for nonzero x,

$$\begin{bmatrix} x_1 & \dots & x_k & 0 & \dots \end{bmatrix}\begin{bmatrix} A_{k\times k} & 0_{k\times(n-k)} \\ 0_{(n-k)\times k} & 0_{(n-k)\times(n-k)} \end{bmatrix}\begin{bmatrix} x_1 \\ \dots \\ x_k \\ 0 \\ \dots \end{bmatrix} = \begin{bmatrix} x_1 & \dots & x_k \end{bmatrix} A_{k\times k}\begin{bmatrix} x_1 \\ \dots \\ x_k \end{bmatrix} > 0$$

By the previous theorem, A is LU-factorizable. Suppose $A = LU$. Since A is symmetric,

$$LU = A = A^T = U^TL^T \implies L^{-1}LU(L^T)^{-1} = L^{-1}U^TL^T(L^T)^{-1} \implies U(L^T)^{-1} = L^{-1}U^T.$$

It follows that $U(L^T)^{-1}$ is upper triangular but $L^{-1}U^T$ is lower triangular. Therefore they must both be some diagonal matrix D. Then $U(L^T)^{-1} = D \implies U = DL^T$ and thus $A = LDL^T$, from which D must also be positive definite (part of HW). Define $\sqrt{D}$ to be the diagonal matrix with each entry being the square root of the corresponding entry in D. Then we can define $\tilde{L}L = L\sqrt{D}$. Then

$$\tilde{L}\tilde{L}^T = L\sqrt{D}\sqrt{D}^TL^T = LDL^T,$$

and the claim of existence follows. Uniqueness: if $A = LL^T = MM^T$ then

$$I = L^{-1}MM^TL^{-T} = (L^{-1}M)(L^{-1}M)^T \implies (L^{-1}M) = (L^{-1}M)^{-T}.$$

Again the LHS is lower triangular and the RHS upper. Therefore both need to be diagonal, and since $(L^{-1}M)(L^{-1}M)^T = I$ the entries must be ± 1. Since $M = L(L^{-1}M)$ one concludes that the entries of M differ from those of L by at most signs. Hence the uniqueness, assuming a positive diagonal. □

Beginning of Feb. 22, 2021

4.3 Pivoting and Constructing an Algorithm

Notice that while we do Gaussian elimination, we are simultaneously doing LU decomposition. By juxtaposing an I on the right of A, i.e., $\begin{bmatrix} A & I \end{bmatrix}$, while we make A upper triangular by using **pivot elements** from **pivot rows** to eliminate all below-the-diagonal entries, $\begin{bmatrix} A & I \end{bmatrix}$ becomes $\begin{bmatrix} U & L^{-1} \end{bmatrix}$. Then $A = LU$.

Theorem 4.3.1

If all the **pivot elements** are nonzero in Gaussian elimination, then $A = LU$.

Pivoting

Why do we need pivoting?

Consider the example $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, where we immediately see $x_1 = x_2 = 1$.

However, if we were to use Gaussian elimination to solve this system, we can't. 1 cannot be eliminated by 0.

Instead, we need to replace 0 by ϵ and solve

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Longrightarrow \begin{bmatrix} \epsilon & 1 \\ 0 & 1-1/\epsilon \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2-1/\epsilon \end{bmatrix}.$$

This gives the solution

$$x_2 = \frac{2-1/\epsilon}{1-1/\epsilon} \approx 1 \text{ and } x_1 = (1-x_2)/\epsilon \approx 0.$$

Clearly this contradicts $x_1 = 1$. This is because before $1 - x_2$ can take place, the computer already treats x_2 as 1 given that $2 - 1/\epsilon \approx 1 - 1/\epsilon$. (For example, `MARC-32` only stores a mantisa up to 7 digits, so if 1 and 2 are too insignificant as compared to $1/\epsilon$, $1 - 1/\epsilon = 1/\epsilon = 2 - 2/\epsilon$, and then $1 - x_2 = 0$. An underflow occurs.)

However, it is not the small ϵ that causes the error. Instead, the error happens because ϵ is too small compared to other elements of the row. For example

$$\begin{bmatrix} 1 & 1/\epsilon \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/\epsilon \\ 2 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1/\epsilon \\ 0 & 1-1/\epsilon \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/\epsilon \\ 2-1/\epsilon \end{bmatrix},$$

where the correct solution is

$$\begin{cases} x_2 = (2-1/\epsilon)/(1-1/\epsilon) \approx 1 \\ x_1 = 1/\epsilon - x_2/\epsilon \approx 0 \end{cases}$$

since both $1/\epsilon$ and x_2/ϵ will be computed as 0 because of "underflow".

The remedy to this problem is by switching the pivot element or switching the order of equations so that we no longer have a pivot that is too small compared to other elements in the row. For example, back to the first problem

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 1-\epsilon \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1-2\epsilon \\ 2 \end{bmatrix}$$

which can be correctly solved by the computer. **Takeaway: $Ax = B \iff PAx = Pb$.**

Gaussian Elimination with Scale Row Pivoting

Suppose we have the permuted linear system $PAx = Pb$. (In computers we don't need an extra step of multiplication. When writing codes, we simply need to specify which entry to use, i.e., instead of using A_{ij} we use $A_{P_i,j}$.)

We begin by computing the **scale** of each row:

$$s_i := \max_{1 \leqslant j \leqslant n} |a_{ij}|,$$

i.e., the largest absolute value of all entries in a row.

Next we **factorize** and pick the pivot row as the row for which $|a_{i,1}|/s_i$ is the largest. Once the first pivot is determined, record it in the permutation array (switching the first entry 1 with the one that corresponds to the first pivot) and do the eliminations on the first column. Then for other remaining rows (we inspect every row but the one that contains the first pivot, i.e., the one corresponding to the first element of the updated array), find the pivot with the largest $|a_{i,2}|/s_i$. So on and so forth.

Beginning of Feb. 24, 2021

Factorizations $PA = LU$

Let $P_1, P_2, \ldots, P_n$ be the indices of the rows in the order in which they become the pivot rows. (We first scale the rows, then switch, switch, and switch).
Meanwhile, Gaussian elimination with scaled row pivoting gives

$$A = A^{(1)} \to A^{(2)} \to \cdots \to A^{(n)},$$

where in each $\to$ we switch two rows (p_i and i) based on the corresponding (elementary) permutation matrix.
To see how $A^{(k+1)}$ is obtained from $A^{(k)}$ where the corresponding P permutes i^{th} and j^{th} rows,

$$a_{p_i j}^{(k+1)} = \begin{cases} a_{pj}^{(k)} & i \leqslant k \text{ or } i > k > j \\ a_{p_i j}^{(k)} - (a_{p_i k}^{(k)} / a_{p_k k}^{(k)}) a_{(p_k j)}^{(k)} & i > k \text{ and } j > k \\ a_{p_i k}^{(k)} / a_{p_k k}^{(k)} & i > k \text{ and } j = k \end{cases}$$

The first case corresponds to the rows that are not altered (because we are done with them already), the second corresponds to the entries that are affected by this process, and the third corresponds to the row that gets scaled.

Theorem 4.3.2

If all pivot elements of the last matrix $A^{(k)}$ are nonzero, then $A = LU$ (or $PA = LU$).

Theorem 4.3.3

Define a permutation matrix $P_{ij} = \delta_{p_i,j}$. Define an upper triangular matrix U whose elements are $u_{ij} = a^{(n)}_{p_i,j}$ if $j \geqslant i$ (and 0 otherwise), and define a unit lower triangular matrix L whose elements are $\ell_{ij} = a^{(n)}_{p_i,j}$ if $j < i$. Then $PA = LU$.

Theorem 4.3.4

If a factorization $PA = LU$ is produced from the Gaussian algorithm with scaled row pivoting, then solving $Ax = b$ is obtained by first solving $Lz = Pb$ and then $Ux = z$.

Operation Counts

Since addition and subtraction take much less time than multiplication and division, we only count the steps, we only count steps involving the latter, which are known as **long operations**.

Next question: how many long operations?

For $A^{(1)} \to A^{(2)}$, we first need to do the factorization process. In determining p_1 (pivot of first row) we need n divisions. Then for the remaining $n-1$ rows, one factor is computed by division and each row involves another $n-1$ computations (as the first one get eliminated to 0, not by division). For these $p-1$ rows, each row involves n ops. Therefore the total is $n + n(n-1) = n^2$ ops from $A^{(1)}$ to $A^{(2)}$.

Similarly, from $A^{(2)}$ to $A^{(3)}$ we are left with a $(n-1) \times (n-1)$ matrix and there are $(n-1)^2$ ops. The total number of ops required in factorization is

$$n^2 + (n-1)^2 + \cdots + 3^2 + 2^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} - 1 \approx \frac{n^3}{3} + \frac{n^2}{2}.$$

Having done the factorization, the solution phase only involves back substitution and there are $1 + 2 + \cdots + n = n^2/2 + n/2$ ops. Since there are two back substitutions the total count is n^2.

Theorem 4.3.5

If Gaussian elimination is used with scaled row pivoting, the solution of the system $Ax = b$ with fixed A and m different vectors b involves approximately

$$\frac{n^3}{3} + (1/2 + m)n^2 \text{ ops.}$$

Beginning of Feb. 26, 2021

Notice that there is no error analysis for these factorizations; there is no error.

Diagonally Dominant Matrices

Definition 4.3.6

A $n \times n$ matrix is **diagonally dominant** if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|,$$

i.e., the diagonal entries are "dominant" enough that it can almost be treated as a diagonal matrix.

Theorem 4.3.7

Gaussian eliminations without pivoting preserves the diagonal dominance of a matrix.

Proof. Notice that it suffices to prove that $A^{(1)} \to A^{(2)}$ preserves diagonal dominance. Since all other elements of the first row except a_{11} become 0 and a_{ij} becomes $a_{ij} - (a_{i1}/a_{11})a_{1j}$. It is equivalent to showing

$$|a_{ii} - (a_{i1}/a_{11})a_{1i}| > \sum_{\substack{j=2 \\ j \neq i}}^{n} |a_{ij} - (a_{i1}/a_{11})a_{1j}|.$$

It suffices to prove the stronger version (by triangle inequality)

$$|a_{ii}| - |(a_{i1}/a_{11})a_{1i}| > \sum [|a_{ij}| + |(a_{i1}/a_{11})a_{1j}|]$$

or

$$|a_{ii}| - \sum_{j \neq 2} |a_{ij}| > \sum_{i=2}^{n} |(a_{i1}/a_{11})a_{1j}|.$$

By the diagonal dominance $|a_{ii}| - \sum_{j \neq i} |a_{ij}| > |a_{i1}|$ so it suffices to prove the RHS $\leqslant |a_{i1}|$. Indeed, this is by Cauchy inequality applied to $\sum |a_{i1}/a_{11}|$ and $\sum |a_{1j}|$. □

Corollary 4.3.8

Every diagonally dominant matrix is nonsingular and has a LU factorization.

Proof. Indeed, each $A^{(n)}$ is diagonally dominant. □

Corollary 4.3.9

There is no need to consider the scaled version of Gaussian elimination if given a diagonally dominant matrix. The pivots elements are already the ones along the diagonal.

Tridiagonal System

Recall that a **tridiagonal** matrix satisfies $a_{ij} = 0$ if $|i - j| > 1$. It is obvious that once we eliminate the lower diagonal then back substitution solves the system.

4.4 Norms (Vectors & Matrices) and Analysis of Errors

Talking about relative errors and "distances between values" in a more general context, we need to introduce the notion of norms. Recall that, on a vector space V, a **norm** is a function $\|\cdot\| : V \to \mathbb{R}_{\geqslant 0}$ that obeys 3 postulates:

(1) non-degeneracy: $\|x\| \geqslant 0$ for all x and $\|x\| = 0$ if and only if $x = 0$,

(2) absolute homogeneity: $\|\lambda x\| = |\lambda|\|x\|$ for $\lambda \in \mathbb{R}$, and

(3) triangle inequality (subadditivity): $\|x + y\| \leqslant \|x\| + \|y\|$.

Usually we deal with $\mathbb{R}^n$ in the context of this course. Recall the various norms: Euclidean norm $\|x\|_2$, the p-norm $\|x\|_p$ and the ∞-norm $\|x\|_\infty$.

Beginning of March 1, 2021

Matrix Norms

First recall that all $n \times n$ matrices form a vector space. Since we are most familiar with $\mathbb{R}^n$, we prefer a matrix norm intimately related to vector norms for $\mathbb{R}^n$.

Definition 4.4.1: Matrix Norms

If a vector norm $\|\cdot\|$ has been specified, the matrix norm **subordinate** to it is defined by

$$\|A\| = \sup\{\|Au\| : u \in \mathbb{R}^n, \|u\| = 1\}.$$

Notice immediately that taking $\|u\| \leqslant 1$ provides the same as $\|u\| = 1$: $\sup_{\|u\| \leqslant 1} \|Au\| \geqslant \sup_{\|u\|=1} \|Au\|$ is trivial, and the converse is because

$$\|u\| < 1 \implies \frac{1}{\|u\|} > 1 \implies \|Au\| < \|A(u/\|u\|)\| = (1/\|u\|)\|Au\|.$$

Theorem 4.4.2

If $\|\cdot\|$ is any norm on $\mathbb{R}^n$, then the subordinate $\|A\|$ defines a norm on the linear space of all $n \times n$ matrices.

Proof. We verify the three criteria:

(1) Non-degeneracy. Nonnegativity is trivial. Now assume $\|A\| = 0$. Clearly if A is the zero matrix then $\|Au\| = 0$. For the converse, suppose A is not the zero matrix. Then some column of A needs to be nonzero. Consider $v = (0, \dots, 0, 1, 0, \dots)^T$ where the 1 corresponds to the nonzero column. Then $Av \neq 0$ and $\|Av\| > 0$ and thus $\sup_{\|u\|=1} \|Au\| \geqslant \|Av\| > 0$.

(2) Absolute homogeneity is trivial.

(3) Subadditivity.

$$\begin{aligned}\|A+B\| &= \sup_{\|u\|=1} \|(A+B)u\| = \sup_{\|u\|=1} \|Au+Bu\| \\ &\leqslant \sup_{\|u\|=1} (\|Au\| + \|Bu\|) \\ &\leqslant \sup_{\|u\|=1} \|Au\| + \sup_{\|v\|=1} \|Bv\| = \|A\| + \|B\|.\end{aligned}$$

□

Proposition 4.4.3

A direct result from the definition is that $\|Ax\| \leqslant \|A\|\|x\|$. *Proof is trivial by normalizing $x/\|x\|$ for nonzero x.*

Theorem 4.4.4

The matrix norm subordinate to $\|\cdot\|_\infty$ is given by

$$\|A\|_\infty = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|,$$

i.e., the largest value possible by taking the $\sum$ of absolute values of entries in a row.

Proof. Assume we are talking with respect to $\|\cdot\|_\infty$. For $\leqslant$, notice that

$$\begin{aligned}\sup_{\|x\|=1} \|Ax\| &= \sup_{\|x\|=1} (\text{“entry (row) of } Ax \text{ with maximum abs value”}) \\ &= \sup_{\|x\|=1} \max_{i\leqslant i\leqslant n} \left| \sum_{j=1}^{n} a_{ij}x_j \right| \leqslant \sup_{\|x\|=1} \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}||x_j| \\ &\leqslant \max_{1\leqslant i\leqslant n} \sum_{i=1}^{n} |a_{ij}| \text{ since } \|x\| = 1 \implies |x_j| \leqslant 1.\end{aligned}$$

In order to prove $\max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}| \leqslant \sup_{\|x\|=1} \|Ax\|$, suppose row $\tilde{i}$ is the one that attains the max. We define

$$x_0 := \begin{bmatrix} \operatorname{sgn}(a_{\tilde{i},1}) & \operatorname{sgn}(a_{\tilde{i},2}) & \cdots & \operatorname{sgn}(a_{\tilde{i},n}) \end{bmatrix}^T$$

then

$$\|Ax_0\| = \max_{1\leqslant i\leqslant n} \left| \sum_{j=1}^{n} a_{ij} \operatorname{sgn}(a_{\tilde{i},j}) \right| \geqslant \left| \sum_{j=1}^{n} a_{\tilde{i},j} \operatorname{sgn}(a_{\tilde{i},j}) \right| = \sum_{j=1}^{n} |a_{\tilde{i},j}| = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|.$$

□

Beginning of March 3, 2021

Remark. The matrix norm subordinate to $\|\cdot\|_1$ is highly similar, except now we are taking the largest possible sum obtained by summing over the absolute values of a column. See HW6.

If we take $\|\cdot\|_2$, it's something different; it relates to the eigenvalues of A, and we'll discuss about that later.

Immediately we see two properties:

(1) $\|I\| = 1$, as $\sup_{\|x\|=1} \|Ix\| = \sup_{\|x\|=1} \|x\| = 1$.

(2) $\|AB\| \leqslant \|A\|\|B\|$, again by definition of supremum:

$$\|AB\| = \sup_{\|x\|=1} \|ABx\| \leqslant \sup_{\|x\|=1} [\|A\|\|Bx\|] \leqslant \sup_{\|x\|=1} [\|A\|\|B\|\|x\|] = \|A\|\|B\|.$$

Recall that we have $\|Ax\| \leqslant \|A\|\|x\|$ simply by normalizing x to $x/\|x\|$.

Condition Number Revisited

Suppose we want to solve $Ax = b$ where $A_{n\times n}$ is invertible. We treat b as input and x as output. Suppose b is pertubed and becomes $\tilde{b}$ so that the solution x becomes $\tilde{x}$. What are their relations?

$$x - \tilde{x} = A^{-1}b - A^{-1}\tilde{b} = A^{-1}(b - \tilde{b}) \implies \|x - \tilde{x}\| = \|A^{-1}(b - \tilde{b})\| \leqslant \|A^{-1}\|\|b - \tilde{b}\|.$$

Recall that the condition number describes the relation between the relative errors, i.e., between $\|x - \tilde{x}\|/\|x\|$ and $\|b - \tilde{b}\|/\|b\|$. Notice that $\|b\| = \|Ax\| \leqslant \|A\|\|x\|$. Multiplying the two inequalities together, we have

$$\|x - \tilde{x}\|\|b\| \leqslant \|A^{-1}\|\|A\|\|x\|\|b - \tilde{b}\| \implies \frac{\|x - \tilde{x}\|}{\|x\|} \leqslant \|A\|\|A^{-1}\|\frac{\|b - \tilde{b}\|}{\|b\|},$$

where $\|A\|\|A^{-1}\|$ is the conditional number of A, denoted $\kappa(A)$. When $\kappa(A)$ is small (in fact, $\|A\|\|A^{-1}\| \geqslant \|AA^{-1}\| = 1$, so $\kappa(A) \geqslant 1$), a small pertubation in $\tilde{b}$ also produces a small perturbation in $\tilde{x}$.

Example 4.4.5. Consider $A = \begin{bmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{bmatrix}$. This seemingly nice matrix is not very friendly:

$$A^{-1} = \frac{1}{\epsilon^2}\begin{bmatrix} 1 & -1-\epsilon \\ -1+\epsilon & 1 \end{bmatrix}$$

which, in $\|\cdot\|_\infty$, gives $\|A\|_\infty = 2 + \epsilon$ and $\|A^{-1}\|_\infty = (2+\epsilon)/\epsilon^2$. Therefore $\kappa(A) = (2+\epsilon)^2/\epsilon^2 \geqslant 4/\epsilon$. If ϵ is small, $\kappa(A)$ is *huge*.

We do not want a large $\kappa(A)$ because we are not only interested in computing $Ax = b$ for one specific b; when trying many different b's it may happen that for some b the ratio between $\|b - \tilde{b}\|/\|b\|$ and $\|x - \tilde{x}\|/\|x\|$ may reach this huge number, which is a trouble.

If we want to solve $Ax = b$ numerically, we obtain an approximate solution $\tilde{x}$. We define the **residue vector** to be $r := b - A\tilde{x}$ and the **error vector** $e := x - \tilde{x}$. Then

$$Ae = Ax - A\tilde{x} = b - A\tilde{x} = r.$$

Theorem 4.4.6

$$\frac{1}{\kappa(A)}\frac{\|r\|}{\|b\|} \leqslant \frac{\|e\|}{\|x\|} \leqslant \kappa(A)\frac{\|r\|}{\|b\|}.$$

Proof. From $Ae = r$ we have $\dfrac{\|e\|}{\|x\|} \leqslant \kappa(A)\dfrac{\|r\|}{\|b\|}$; this is nothing but $\dfrac{\|x - \tilde{x}\|}{\|x\|} \leqslant \kappa(A)\dfrac{\|b - \tilde{b}\|}{\|b\|}$. For the other direction,

notice that $Ae = r$ can be written as $A^{-1}r = e$ since A is assumed to be invertible. Then

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

Notice that $\kappa(A^{-1}) = \|A^{-1}\|\|(A^{-1})^{-1}\| = \kappa(A)$. □

4.5 Neumann Series and Iterative Refinement

Let $(V, \|\cdot\|)$ be a normed (vector) space. Our goal is to find a convergent sequence of vectors $v^{(1)}, v^{(2)}, \dots$ that eventually converge to our desired solution, v. (Recall that $v^{(k)} \to v$ if $\lim_{k\to\infty} \|v^{(k)} - v\| = 0$.

Remark. Recall that if V is a finite-dimensional vector space, then all norms on V are equivalent. Therefore if $\lim_{k\to\infty} \|v^{(k)} - v\| \to 0$ in some norm, the same holds for any other norm.

Remark. Also, recall that finite-dimensional normed vector spaces are complete; if $\dim(V) = n$ then $V \equiv \mathbb{R}^n$, isometrically isomorphic. Therefore it makes sense to use an alternate criterion for convergence, the **Cauchy criterion**: convergent if $\lim \|v^{(m)} - v^{(n)}\| \to 0$ as $\min(m,n) \to \infty$.

Beginning of March 5, 2021

Theorem 4.5.1

If A is an $n \times n$ matrix such that $\|A\| < 1$, then $I - A$ is invertible and

$$(I-A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

Proof. We first show that $I - A$ is invertible. Suppose for contradiction that $I - A$ is not invertible. Then for some $x \neq 0$ we have $(I - A)x = 0$. WLOG assume $\|x\| = 1$; we have

$$(I-A)x = 0 \implies Ax = x \implies 1 = \|x\| = \|Ax\| \leqslant \|A\|\|x\| = \|A\|,$$

contradiction. Now we show that the partial sums $\sum_{k=0}^{m} A^k$ converge to $(I-A)^{-1}$ (so that the infinite sum is indeed its inverse). This is equivalent to showing

$$(I-A)\sum_{k=0}^{m} A^k \to I.$$

Indeed,

$$(I-A)\sum_{k=0}^{m} A^k = \sum_{k=0}^{m}(A^k - A^{k+1}) = I - A^{m+1} \to I$$

as $\|A^{m+1}\| < \|A\|^{m+1} \to 0$ and so $\|I - (I - A^{m+1})\| = \|A^{m+1}\| \to 0$. □

Remark. *Notice that we do not need to specify which norm $\|A\| < 1$ and the representation of $(I-A)^{-1}$ does not depend on the norm we pick. Therefore we can pick any norm in the first place to work with. There is no guarantee that if some norm works, then all other norm works though.*

This representation for $(I-A)^{-1}$ is enormously helpful as it allows us to bypass the $\mathcal{O}(n^3)$ ops involved in directly computing $(I-A)^{-1}$. When n is very large, picking an appropriate $m \ll n$ and computing $\sum_{k=0}^{m} A^k$ involves $m \cdot \mathcal{O}(n^2)$ steps without losing too much accuracy by omitting $\sum_{k=m+1}^{\infty} A^k$.

Example 4.5.2. Use the Neumann series to compute the inverse of

$$B = \begin{bmatrix} 0.9 & -0.2 & -0.3 \\ 0.1 & 1.0 & -0.1 \\ 0.3 & 0.2 & 1.1 \end{bmatrix}.$$

Solution

Let $B = I - A$ so

$$A = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ -0.1 & 0.0 & 0.1 \\ -0.3 & -0.2 & -0.1 \end{bmatrix}.$$

Indeed, $\|A\|_\infty < 1$ so the theorem applies: indeed, the sum of absolute values of the entries in the first row is $0.6 < 1$ and the same holds for other two. Computations are omitted.

Theorem 4.5.3

If A and B are two matrices such that $\|I - AB\| < 1$, then A and B are invertible. Furthermore,

$$A^{-1} = B\sum_{k=0}^{\infty}(I-AB)^k \text{ and } B^{-1} = A\sum_{k=0}^{\infty}(I-AB)^k.$$

Proof. Applying the previous theorem, we have

$$(AB)^{-1} = \sum_{k=0}^{\infty}(I-AB)^k,$$

so

$$A^{-1} = BB^{-1}A^{-1} = B(AB)^{-1} = B\sum_{k=0}^{\infty}(I-AB)^k$$
$$B^{-1} = B^{-1}A^{-1}A = (AB)^{-1}A = \sum_{k=0}^{\infty}(I-AB)^k \cdot A.$$

□

Remark. In some situations, this theorem allows us to generalize the previous one. For example, consider

$$A = \begin{bmatrix} 1000 & 1 & 2 \\ 0 & -100 & 3 \\ 0 & 0 & 1000 \end{bmatrix} \implies I - A = \begin{bmatrix} -999 & -1 & -2 \\ 0 & 101 & -3 \\ 0 & 0 & -999 \end{bmatrix}.$$

Clearly $\|I - A\|_\infty > 0$. We can introduce $B = \texttt{diag(1/2000, 1/200, 12000)}$ so that $\|AB\|_\infty < 1$, and then we can compute A^{-1}.

Beginning of March 8, 2021

Often times we cannot immediately find $B := A^{-1}$, but we can find $B \approx A^{-1}$. Then the above way indeed gives a nice way to calculate / approximate A^{-1}.

Iterative Refinement

If $x^{(0)}$ is an approximate solution to $Ax = b$. Then the precise solution is

$$e = x^{(0)} + A^{-1}(b - Ax^{(0)}) := x^{(0)} + e^{(0)}$$

where $e^{(0)}$ denotes the **error vector**. We define $r^{(0)} := b - Ax^{(0)}$ to be the **residual vector**. Then $Ae^{(0)} = r^{(0)}$. To solve this equation, we obtain another approximation $x^{(1)} = x^{(0)} + e^{(0)}$. In other words, if B is an approximation of A^{-1}, we define

$$x^{(0)} = Bb \text{ and } x^{(k+1)} = x^{(k)} + B(b - Ax^{(k)}).$$

Theorem 4.5.4

If $\|I - AB\| < 1$, then the method of iterative refine given by the equation above produces the sequence of vectors

$$x^{(m)} = B\sum_{k=0}^{m}(I - AB)^k b.$$

Then by the previous theorems, $x^{(m)}$ converges to x.

Proof. Recall that $x^{(k+1)} = x^{(k)} + B(b - Ax^{(k)})$. The base case is true since

$$x^{(1)} = x^{(0)} + Bb - BAx^{(0)} = (I - BA)x^{(0)} + Bb.$$

For the inductive step, assuming case m holds,

$$\begin{aligned} x^{(m+1)} &= x^{(m)} + B(b - Ax^{(m)}) \\ &= B\sum_{k=0}^{m}(I - AB)^k b + Bb - BAB\sum_{k=0}^{m}(I - AB)^{(k)} b \\ &= Bb + B(I - AB)\sum_{k=0}^{m}(I - AB)^k b \\ &= B\sum_{k=0}^{m+1}(I - AB)^k b. \end{aligned}$$

For convergence,

$$\|x^{(m)} - x\| = \left\|\left[B\sum_{k=0}^{m}(I - AB)^k - A^{-1}\right]b\right\| \leqslant \left\|B\sum_{k=0}^{m}(I - AB)^k - A^{-1}\right\|\|b\| \to 0.$$

□

4.6 Solution of Equations by Iterative Methods

Recall the Gaussian eliminations and its variants. They are called **direct** methods because we are able to obtain the completely accurate solution after a finite number of steps (scaling, elimination, etc.).
The counterpart, **indirect** methods, uses iteration of a single process to generate a sequence that ideally converges to the solution. The algorithm is stopped when the approximation is close enough to the exact solution.

Some circumstances in which indirect methods are preferred:

(1) large linear systems (n large), where there is too much computation to carry out by direct methods,

(2) **sparse** systems, in which there is a large proportion of zeros in the system, and

(3) solutions of PDEs, where each row of A is discarded after being used.

(4) for a singular matrix, Gaussian elimination will run into problems (dividing by 0) and halt, but iterative methods are more stable: at least it will provide one solution among the infinitely many others.

Example 4.6.1. Consider the simple linear system

$$\begin{bmatrix} 7 & -6 \\ -8 & 9 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}.$$

Recall the fixed point argument $F(x^{(k)}) = x^{(k+1)}$. If $\{x^{(k)}\}$ converges then it will converge to a fixed point of F. Here we adapt a similar methodology.

Jacobi method solves the i^{th} for the i^{th} unknown as follows:

$$\begin{cases} x_1^{(k)} = 6x_2^{(k-1)}/7 + 3/7 \\ x_2^{(k)} = 8x_1^{(k-1)}/9 - 4/9 \end{cases} \implies \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} := F\begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \end{bmatrix} = \begin{bmatrix} 6x_2^{(k-1)}/7 + 3/7 \\ 8x_1^{(k-1)}/9 - 4/9 \end{bmatrix}.$$

Then, if $\left[x_1^{(k)}, x_2^{(k)}\right]^T$ converges, it must converge to a fixed point of F, namely where

$$\begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} 6\tilde{x}_2/7 + 3/7 \\ 8\tilde{x}_1/9 - 4/9 \end{bmatrix} \iff \begin{bmatrix} 7 & -6 \\ -8 & 9 \end{bmatrix}\begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}.$$

Beginning of March 10, 2021

The **Gauss-Seidel** method is similar:

$$x_1^{(k)} = \frac{6}{7}x_2^{(k-1)} + \frac{3}{7} \text{ and } x_2^{(k)} = \frac{8}{9}x_1^{(k)} - \frac{4}{9}.$$

It simply took advantage of $x_1^{(k)}$ when computing $x_2^{(k)}$.

Basic Concepts

Still, consider $Ax = b$. We can split $A = Q + (A - Q)$, i.e., **splitting** it, so the original equation becomes

$$Qx = (Q - A)x + b.$$

Then we use the iterative method to compute

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

The Q we choose needs to satisfy the following:

(1) the sequence $x^{(k)}$ can be easily computed, and

(2) the sequence $x^{(k)}$ converges rapidly to a solution.

Assuming both A and Q are nonsingular, we may multiply both sides by Q^{-1} and obtain the following form which is indeed an iterative process:

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

With the assumption on invertibility, the two forms are equivalent. The latter is more convenient for theoretical analysis but it is always the first form that is used by computers. The computers don't want to compute an extra Q^{-1}!

Theorem 4.6.2

If $\|I - Q^{-1}A\| < 1$ for some subordinate norm, then the sequence produced by above converges to the solution $Ax = b$ for any initial vector $x^{(0)}$.

Order of convergence. We write the error as

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

Then,

$$e^{(k)} = (I - Q^{-1}A)(x^{(k-1)} - x) + Q^{-1}b - x + (I - Q^{-1}A)x = (I - Q^{-1}A)e^{(k-1)} + Q^{-1}b - Q^{-1}Ax$$

where the last two terms cancel, assuming x is the solution. Therefore,

$$e^{(k)} = (I - Q^{-1}A)e^{(k-1)}.$$

Taking the norm:

$$\|e^{(k)}\| = \|(I - Q^{-1}A)e^{(k-1)}\| \leqslant \|I - Q^{-1}A\|\|e^{(k-1)}\|$$

so the convergence is linear (first order) and $\|e^{(k)}\| \leqslant \|I - Q^{-1}A\|^{(k)}\|e^{(0)}\|$. Therefore, the smaller $\|I - Q^{-1}A\|$ the faster the convergence.

The **Richardson Method** simply takes $Q = I$, namely

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

The **Jacobi Method** takes $Q = D$, diagonal of A. Then

$$Dx^{(k)} = (D - A)x^{(k-1)} + b \implies x^{(k)} = (I - D^{-1}A)x^{(k-1)} + D^{-1}b.$$

Theorem 4.6.3

If A is diagonally dominant, then the sequence produced by the Jacobi iteration converges to the solution $Ax = b$ for any starting vector.

Proof. Notice that $D^{-1}A$ is simply another diagonally dominant matrix but with diagonal entries 1. Then $I - D^{-1}A$ simply removes all those 1's along the diagonal. Now consider the $\|\cdot\|_\infty$ of $I - D^{-1}A$. Since $D^{-1}A$ is diagonally dominant, the sum of any entries in a row, excluding the diagonal entry, is always < 1 (since the diagonal entry is 1). Then $\|I - Q^{-1}A\|_\infty < 1$ and convergence follows. □

Beginning of March 15, 2021

Analysis & More Concepts

The next goal is to consider the generalized case

$$x^{(k)} = Gx^{(k-1)} + c$$

for any given G and vector c. Indeed, this is a generalization of what is discussed above if we simply set $G := I - Q^{-1}A$ and $c = Q^{-1}b$. The natural question that arises is, what are the necessary and sufficient conditions on G that ensures a convergence of approximations regardless of starting vector? Some preliminaries first.

Definition 4.6.4

Let A be a matrix. We say λ is an **eigenvalue** of A if $\det(A - \lambda I) = 0$.

Definition 4.6.5

The **spectral radius** of A is $\rho(A) = \max|\lambda|$. (If λ is complex, take the modulus.) Intuitively, $\rho(A)$ is the radius of the smallest circle in the complex plane that contains all eigenvalues of A.

Definition 4.6.6

Let A, B be matrices. We say A and B are **similar** if there exists S such that $S^{-1}AS = B$. It follows that A and B have the same eigenvalues (not necessarily the same eigenvectors, of course). Indeed,

$$\begin{aligned} 0 = \det(B - \lambda I) &= \det(S^{-1}AS - \lambda I) \\ &= \det(S^{-1}(A - \lambda I)S) \\ &= \det(S^{-1})\det(S)\det(A - \lambda I) \implies \det(A - \lambda I) = 0. \end{aligned}$$

Usually, we define **spectrum** of A, written $\Lambda(A)$, to be the set of all eigenvalues of A. Therefore if $A \sim B$ then $\Lambda(A) = \Lambda(B)$. In fact, $\sim$ is reflexive, symmetric, and transitive.

Lemma 4.6.7

Every square matrix is similar to an upper triangular matrix whose off-diagonal elements are arbitrarily small.

Proof. Let A be given. We want to show that $A \sim D + \epsilon U$. *Schur's Theorem* (to be covered later) states that $A \sim T$ where T is an upper triangular matrix, real or complex. Let $\epsilon > 0$ be given, assuming $\epsilon < 1$. Then consider the diagonal matrix and its inverse as given below:

$$D = \text{diag}(\epsilon, \epsilon^2, \dots, \epsilon^n) \text{ and } D^{-1} = \text{diag}(\epsilon^{-1}, \epsilon^{-2}, \dots, \epsilon^{-n}).$$

Now we consider $D^{-1}TD$. $D^{-1}T$ multiplies the rows of T and $(D^{-1}T)D$ multiplies the columns by D. Then

$$T_{i,j} \mapsto T_{i,j} \cdot \epsilon^{j-i} \implies T_{i,j} \mapsto \begin{cases} T_{i,j} & i = j \\ \leqslant \epsilon T_{i,j} & i < j. \end{cases}$$

This finishes the proof. □

Theorem 4.6.8

The spectral radius function satisfies

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

where the infinum is taken over all subordinate matrix norms.

Proof. To show $\rho(A) \geqslant \inf_{\|\cdot\|} \|A\|$, by the lemma above, $A \sim D + \epsilon U$. Then $\rho(A) = \rho(D + \epsilon U)$ as they have exactly the same eigenvalues. Notice that if $A \sim B$ then any subordinate norm of A is equal to the some (other or not) subordinate norm of B:

$$\|A\| = \|B\|' := \|PBP^{-1}\| \text{ where } A = P^{-1}BP.$$

Indeed, $\|B\|' = \|P^{-1}BP\| = \sup_{\|x\|=1} \|P^{-1}BPx\| = \sup_{\|P^{-1}y\|=1} \|P^{-1}By\| = \sup_{\|y\|'=1} \|By\|'$, where the third = involves substituting $y = Px$ and the fourth involves a new vector norm $\|y\|' = \|P^{-1}y\|$.
Suppose that $\|C\|_\infty = c$ for some constant c. Then,

$$\|D + \epsilon U\|_\infty \leqslant \max\{|\lambda_1|, \cdots, |\lambda_n|\} + c\epsilon.$$

Oh the other hand,

$$\rho(D + \epsilon U) = \max_{\lambda \in \Lambda} |\lambda| = \max\{|\lambda_1|, \cdots, |\lambda_n|\}.$$

Therefore,

$$\rho(D + \epsilon U) = \rho(A) \geqslant \|D + \epsilon U\|_\infty - c\epsilon.$$

Notice that $\|D+\epsilon U\|_\infty$ can be re-written as some other subordinate norm $\|A\|'$ by what is shown above. Therefore

$$\rho(A) \geqslant \|A\|' - c\epsilon \text{ for all } \epsilon > 0 \implies \rho(A) \geqslant \inf_{\|\cdot\|} \|A\|.$$

Beginning of March 17, 2021

Now it remains to show that $\rho(A) \leqslant \inf \|A\|$. This is equivalent to showing that $\rho(A) \leqslant \|A\|$ for any $\|\cdot\|$. Indeed, if $\lambda \in \Lambda$ then

$$|\lambda|\|x\| = \|\lambda x\| = \|Ax\| \leqslant \|A\|\|x\|$$

and the claim follow from the fact that $\lambda \leqslant \rho(A)$ and taking infimum.

Alternatively, suppose for contradiction that $\rho(A) > \|A\|$. Then there exists some eigenvalue λ_∞ and (nonzero) eigenvector x_∞ where $|\lambda_\infty| > \|A\|$. This gives

$$\|Ax_\infty\| = \|\lambda_\infty x_\infty\| = |\lambda_\infty|\|x_\infty\|.$$

Normalizing this gives

$$\|A(x_\infty/\|x_\infty\|)\| = |\lambda_\infty| \geqslant \|A\| = \sup_{\|x\|=1} \|Ax\|,$$

contradiction. Therefore $\rho(A) \leqslant \inf \|A\|$. □

Recall that when spilling we used the method

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

Theorem 4.6.9

For the iteration formula

$$x^{(k)} = Gx^{(k-1)} + c,$$

to produce a sequence converging to $(I-G)^{-1}c$, for any starting vector $x^{(0)}$, it is necessary and sufficient that $\rho(G) < 1$. *Notice that $(I-G)^{-1}c$ is the fixed point of $F(x) = G(x) + c$.*

Proof. We show $\rho(G) < 1 \iff x^{(k)} \to (I-G)^{-1}c$.
$\implies$ is obvious: if $\rho(G) < 1$ then some norm gives $\|G\| < 1$. Then the result follows from Theorem 4.5.1.
For $\impliedby$, assume $\rho(A) \geqslant 1$. If $\rho(A) > 1$ then some λ has $|\lambda > 1|$. Let $x^{(0)} = x_\lambda$ the eigenvector. Then

$$\begin{aligned} x^{(0)} &= x_\lambda \\ x^{(1)} &= Gx^{(0)} + c = \lambda x_\lambda + c \\ x^{(2)} &= G^2x^{(0)} + Gc + c = \lambda^2 x_\lambda + \dots \end{aligned}$$

It follows that $x^{(n)}$ does not converge. Similarly, if $\rho(A) = 1$, we again get a divergent series. □

Corollary 4.6.10

The iteration formula $Qx^{(k)} = (Q-A)x^{(k-1)} + b$ will produce a sequence converging to the solution of $Ax = b$ for any $x^{(0)}$ if $\rho(I - Q^{-1}A) < 1$.

Gauss-Seidel Method

Theorem 4.6.11

If A is diagonally dominant, then the Gauss-Seidel method

$$Qx^{(k)} = (Q-A)x^{(k-1)} + b \text{ with } Q = \text{lower triangular part of } A$$

converges for any starting vector.

Proof. By the above corollary, it suffices to check that $\rho(I - Q^{-1}A) < 1$. Indeed, let λ be any eigenvalue of $I - Q^{-1}A$ and let x be the corresponding eigenvector. WLOG assume $\|x\|_\infty = 1$. Then

$$(I - Q^{-1}A)x = \lambda x \implies Qx - Ax = Q\lambda x = \lambda Q x.$$

We rewrite $A = L + D + U$ where L is purely lower triangular, D diagonal, and U purely upper triangular. Then

$$(Q - A)x = -Ux = \lambda(L + D)x.$$

Since U is purely upper triangular and $D + L$ lower triangular, the computation can be simplified into

$$-\sum_{j=i+1}^{n} a_{i,j}x_j = \lambda \sum_{j=1}^{i} a_{i,j}x_j \qquad 1 \leqslant i \leqslant n.$$

Since A is diagonally dominant, it's natural to leave all diagonal terms on one side and everything else the other:

$$\lambda a_{ii}x_i = -\lambda \sum_{j=1}^{i-1} a_{i,j}x_j - \sum_{j=i+1}^{n} a_{i,j}x_j \qquad 1 \leqslant i \leqslant n.$$

Recall the assumption is that $\|x\|_\infty = 1$, for some i we have $|x_i| = 1$. Fix this i and it follows that $|x_j| \leqslant 1$ for all other j's. Then,

$$|\lambda||a_{i,i}| \leqslant |\lambda| \sum_{j=1}^{i-1} |a_{i,j}||x_j| + \sum_{j=i+1}^{n} |a_{i,j}||x_j| \qquad \text{by a bunch of triangle inequalities}$$

and since other $|x_k|$'s are no more than 1,

$$|\lambda||a_{i,i}| \leqslant |\lambda| \sum_{j=1}^{i-1} |a_{i,j}| + \sum_{j=i+1}^{n} |a_{i,j}|.$$

Recall that A is diagonally dominant! Dividing the above by $|\lambda|$ gives

$$|a_{i,i}| \leqslant \sum_{j=1}^{i-1} |a_{i,j}| + \frac{1}{|\lambda|} \sum_{j=i+1}^{n} |a_{i.j}|,$$

which gives a contradiction unless $|\lambda| < 1$ for all λ. This finishes the proof. □

Beginning of March 22, 2021

Iterative Matrices

Recall that Ricardson uses $x^{(k)} = (I - Q^{-1}A)x^{(k-1)} + Q^{-1}b$ where $Q = I$. Jacobi uses $Q = \texttt{diag}(A)$, and Gauß-Seidel uses Q = lower triangular part (including diagonal) of A.

Extrapolation

Extrapolation is a general technique that can be used to improve the convergence properties of a linear iterative process. Again, consider

$$x^{(k)} = Gx^{(k-1)} + c.$$

To improve it, we introduce a parameter $\gamma \neq 0$ and consider instead

$$x^{(k)} = \gamma(Gx^{(k-1)} + c) + (1-\gamma)x^{(k-1)}.$$

A fixed point for above is $x = \gamma(Gx + c) + (1-\gamma)x \implies \gamma x = \gamma(Gx + c)$ so indeed it gives the same result. If we define

$$G_\gamma := \gamma G + (1-\gamma)I,$$

we have

$$x^{(k)} = G_\gamma x^{(k-1)} + \gamma c.$$

Thus, we have introduced a new iterative method which yields the same solutions. For the original one, we needed to consider $\rho(G)$ to determine if there is convergence (if $\rho(G) < 1$ then convergent follows). Here we instead consider $\rho(G_\gamma) = \rho(\gamma G + (1-\gamma)I)$. Why is this useful though?
Assuming $\rho(G) < 1$, can we find a γ that makes $\rho(G_\gamma)$ even smaller and thus even better? Recall that the smaller ρ is, the faster the convergence is (recall it's always first-order; the convergence is related to $\|G\|^n$).

Lemma 4.6.12

If λ is an eigenvalue of A and if p is a polynomial, then $p(\lambda)$ is an eigenvalue of $p(A)$. *Clear enough.*

Theorem 4.6.13

If the only information available about the eigenvalues of G is that they lie in the interval $[a,b]$ and if $1 \notin [a,b]$, then the best choice of γ is $\gamma := 2/(2-a-b)$. With this γ, we can guarantee that $\rho(G_\gamma) < 1 - |\gamma|d$ where d is the distance from 1 to $[a,b]$.

Example 4.6.14. For example, if all eigenvalues of G are in $[100, 200]$. Then $\rho(G) \geqslant 100$. The iterative method will not work as $\rho(G)$ is too big (need < 1)! However, if we define G_γ as stated in the theorem above,

$$\gamma = \frac{2}{2-100-200} = -\frac{1}{149},$$

then $G_\gamma = \gamma G - (1-\gamma)I \implies \rho(G_\gamma) \leqslant -1 - 99|\delta| = 50/149$. This is *much, much* smaller! The same holds even for G with $\rho(G) < 1$, in which case we will get $\rho(G_\gamma)$ which is even smaller.

Proof of Theorem. Notice that if λ is an eigenvalue of G, then $\gamma\lambda + 1 - \gamma$ is one for G_γ by the lemma above. (Treat $\gamma G + (1-\gamma)I$ as a first-degree polynomial.) Then,

$$\rho(G_\gamma) = \max_{\lambda' \in \Lambda(G_\gamma)} |\lambda'| = \max_{\lambda \in \Lambda(G)} |\gamma\lambda + 1 - \gamma| \leqslant \max_{a \leqslant \lambda \leqslant b} |\gamma\lambda + 1 - \gamma|.$$

The rest is just tedious computation and thus omitted. □

Remark. This theorem shows that it suffices to know the range of the eigenvalues in order to improve the iteration method. In the next chapter we will derive some techniques to actually find the regions in which the eigenvalues lie in.

Chebyshev Acceleration

Suppose again that the iteration is given by

$$x^{(k)} = Gx^{(-1)} + c.$$

Previously, when computing $x^{(k)}$ we only rely on $x^{(k-1)}$. Sure, this works, but is it the most efficient one? In particular, since we have computed all of $x^{(1)}, \dots, x^{(k)}$, can we make use of them as well? Is there a linear combination of these previous terms that gives a better approximation than $x^{(k)}$? To put into mathematical language, we seek

$$u^{(k)} = \sum_{i=0}^{k} a_i^{(k)} x^{(i)}, \text{ where } a_0^{(k)} + \cdots + a_k^{(k)} = 1.$$

Compare this with extrapolation

$$\gamma(Gx^{(k-1)} + c) + (1-\gamma)x^{(k-1)}$$

which uses a linear combination of two of the previous $x^{(n)}$'s. We know that when this is done properly we have a better approximation, so it is natural to wonder if we can make this even better if we use more terms.

Analysis

Similar to before, we compute the error:

$$\begin{aligned} u^{(k)} = \sum_{i=0}^{k} a_i^{(k)} x^{(i)} - x &= \sum_{i=0}^{k} \left[a_i^{(k)} (x^{(i)} - x) \right] \\ &= \sum_{i=0}^{k} \left[a_i^{(k)} G^i (x^{(0)} - x) \right] \\ &= P(G)(x^{(0)} - x) \end{aligned}$$

where P is a polynomial defined by $P(z) = \sum_{i=0}^{k} a_i^{(k)} z^i$. Therefore,

$$\|u^{(k)} - x\| \leqslant \|P(G)\| \cdot \|x^{(0)} - x\|.$$

Again, there is no restriction on which matrix norm we pick. Notice that

$$\rho(P(G)) = \max_{1 \leqslant i \leqslant n} |P(\mu_i)| \leqslant \max_{z \in S} |P(z)|$$

where μ_i are eigenvalues of G and S is the region that bounds all eigenvalues of G. Our goal is to minimize the last expression subject to a constraint $\sum_{i=0}^{k} a_i = 1$, i.e., $P(1) = 1$.

Beginning of March 24, 2021

It turns out that, this is a standard problem in approximation theory and we can indeed find explicit solutions for $S = [a, b]$. We will talk about this later when we get to approximation theory.

One typical problem is if $[a, b] = [-1, 1]$. Then the Chebyshev polynomial T_k $(k \geqslant 1)$ is the unique polynomial of degree k that minimizes the expression

$$\max_{-1 \leqslant z \leqslant 1} |T_k(z)| : \begin{cases} T_0(z) = 1, T_1(z) = z \\ T_k(z) = 2zT_{k-1}(z) - T_{k-2}(z) \quad (k \geqslant 2) \end{cases}$$

For a general $[a, b]$, we only need to scale and translate $[-1, 1]$ if needed.

Chapter 5

Other Topics

5.1 Matrix Eigenvalue Problem: Power Method

Here we primarily focus on how to compute eigenvalues of a matrix using the power method. First, some preliminaries.

Definition 5.1.1

The **conjuage** and **modulus** of $\gamma = \alpha + \beta i \in \mathbb{C}$ is given by

$$\overline{\gamma} = \alpha - \beta i \text{ and } |\gamma| = \sqrt{\alpha^2 + \beta^2}, \text{ respectively.}$$

It is easy to check that $|\gamma|^2 = \gamma\overline{\gamma}$.

Definition 5.1.2

The **inner product** of $x, y \in \mathbb{C}^n$ are given by

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i \overline{y_i} = \overline{y^T} x := y^* x.$$

The norm induced by this product is

$$\|x\|_2 = \sqrt{\langle x, x \rangle} = \sum_{i=1}^{n} |x_i|^2.$$

Theorem 5.1.3: Fundamental Theorem of Algebra, FToA

Every non-nonstant polynomial of degree n with complex coeffieicnts has n roots (not necessarily distinct).

$$P(z) = a_k \prod_{i=1}^{k} (z - z_i).$$

How to compute the eigenvalues and eigenvectors of a matrix?

Recall that if $Ax = \lambda x$ has a nontrivial solution then λ is an eigenvalue of A. The corresponding x is an eigenvector.

Proposition 5.1.4

The following are equivalent:

(1) $A - \lambda I$ maps some nonzero vector into 0,

(2) $A - \lambda I$ is singular, and

(3) $\det(A - \lambda I) = 0$. This cries out for the **direct method** of computing and finding the roots of the characteristic polynomial! It's easy to see that once the dimension of A gets large, this becomes a nightmare!

Power Method

The power method is a method capable of computing eigenvalues and eigenvectors simultaneously. Unfortunately, there are several conditions that need to be imposed on A (otherwise we need variants of power method):

(1) there is a single eigenvalue of maximum modulus, and

(2) there is a linearly independent set of n eigenvectors.

Namely, if $\lambda_1, \dots, \lambda_n$ are eigenvectors, then after relabeling we must have

$$|\lambda_1| > |\lambda_2| \geqslant |\lambda_3| \geqslant \cdots \geqslant |\lambda_n|.$$

The second assumption says there exists a (linearly independent) basis $\{u^{(1)}, \dots, u^{(n)}\}$ (of $\mathbb{C}^n$) such that $u^{(i)}$ is an eigenvector of λ_i.
Now we let $x^{(0)}$ be any nonzero element of $\mathbb{C}^n$ such that it can be expressed as a linear combination of these basis vectors with nonzero coefficient of $u^{(1)}$, i.e.,

$$x^{(0)} = \sum_{i=1}^{n} a_i u^{(i)}, \qquad a_1 \neq 0.$$

Now we define an iterative method by $x^{(k)} = Ax^{(k-1)} = A^k x^{(0)}$. Then,

$$\begin{aligned}
x^{(k)} &= A^k x^{(0)} \\
&= A^k (a_1 u^{(1)} + a_2 u^{(2)} + \cdots + a_n u^{(n)}) \\
&= a_1 A^k u^{(1)} + a_2 A^k u^{(2)} + \cdots + a_n A^k u^{(n)} \\
&= a_1 \lambda_1^k u^{(1)} + a_2 \lambda_2^k u^{(2)} + \cdots + a_n \lambda_n^k u^{(n)} \\
&= \lambda_1^k \left[a_1 u^{(1)} + \left(\frac{\lambda_2}{\lambda_1}\right)^k a_2 u^{(2)} + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^k a_n u^{(n)} \right] \\
&= \lambda_1^k (a_1 u^{(1)} + \epsilon^k) \to \lambda_1^k a_1 u^{(1)}.
\end{aligned}$$

Therefore, $x^{(k)}/\lambda_1^k \to a_1 u^{(1)}$.

Beginning of March 26, 2021

Notice that

$$\lim_{k\to\infty}\frac{x^{(k+1)}}{x^{(k)}}=\lim_{k\to\infty}\frac{\lambda_1^{k+1}(a_1u^{(1)}u^{(1)}+\epsilon^{(k)})}{\lambda_1^k(a_1u^{(1)}+\epsilon^{(k)})}\to\lambda_1.$$

We here introduce the notion of **linear functional** which is a mapping from vectors to scalars ($\mathbb{C}$ or $\mathbb{R}$), i.e., it satisfies

$$\varphi(\alpha x+\beta y)=\alpha\varphi(x)+\beta\varphi(y)\text{ for all }\alpha,\beta\in\mathbb{C}.$$

Then, by the linearity above, for any linear functional,

$$\varphi(x^{(k)})=\lambda_1^k\left[a_1\varphi(u^{(1)})+\varphi(e^{(k)})\right]$$

so

$$\lim_{k\to\infty}r_k:=\lim_{k\to\infty}\frac{\varphi(x^{(k+1)})}{\varphi(x^{(k)})}=\lambda_1\left[\frac{a_1\varphi(u^{(1)})+\varphi(\epsilon^{(k+1)})}{a_1\varphi(u^{(1)})+\varphi(\epsilon^{(k)})}\right]\to\lambda_1.$$

This procedure is called the **power method**. The way to pick φ is of course not unique, as we will discuss more soon.

Aitken Acceleration

If the ratios are regarded as approximations to λ_1, then it makes sense to estimate $|r_k-\lambda_1|$ which should converge to 0. Since

$$\begin{aligned}r_{k+1}-\lambda_1&=\frac{\varphi(x^{(k+2)})-\lambda_1\varphi(x^{(k+1)})}{\varphi(x^{(k+1)})}=\frac{\varphi(x^{(k+2)}-\lambda_1x^{(k+1)})}{\varphi(x^{(k+1)})}\\&=\frac{\varphi\left[\lambda_1^{k+1}(a_1u^{(1)}+\epsilon^{(k+2)})-\lambda_1^{k+2}(a_1u^{(1)}+\epsilon^{(k+1)})\right]}{\varphi(\lambda_1^{k+1}(a_1u^{(1)}+\epsilon^{(k+1)}))}\\&=\frac{\lambda_1^{k+2}\varphi(\epsilon^{(k+2)}-\epsilon^{(k-1)})}{\lambda_1^{k+1}\varphi(a_1u^{(1)}+\epsilon^{(k+1)})}\\&=\lambda_1\frac{\varphi(\epsilon^{(k+2)}-\epsilon^{(k+1)})}{\varphi(a_1u^{(1)}+\epsilon^{(k+1)})}.\end{aligned}$$

Therefore

$$\frac{r_{k+1}-\lambda_1}{r_k-\lambda_1}=\frac{\varphi(a_1u^{(1)}+\epsilon^{(k)})}{\varphi(\epsilon^{(k+1)}-\epsilon^{(k)})}\cdot\frac{\varphi(\epsilon^{(k+2)}-\epsilon^{(k-1)})}{\varphi(a_1u^{(1)}+\epsilon^{(k+1)})}=c+\delta_k$$

for some $|c|<1$ and $\delta_k\to0$.

Based on this information, we can do a general procedure known as the **Aitken acceleration**:

Theorem 5.1.5

Let $\{r_n\}$ be a sequence of numbers that converges to a limit r. Then the new sequence

$$s_n:=\frac{r_nr_{n+2}-r_{n+1}^2}{r_{n+2}-2r_{n+1}+r_n},\quad n\geqslant0$$

converges to r faster if $r_{n+1}-r=(c+\delta_n)(r_n-r)$ with $|c|<1$ and $\delta_n\to0$, i.e., $\lim_{n\to\infty}\dfrac{s_n-r}{r_n-r}=0$.

Proof. Let $h_n:=r_n-r$ for all n, i.e., the error sequence of r_n. Then immediately

$$s_n=\frac{(r+h_n)(r+h_{n+2})-(r+h_{n+1})^2}{(r+h_{n+2}-2(r+h_{n-1})+(r+h_n)}=r+\frac{h_nh_{n+2}-h_{n+1}^2}{h_{n+2}-2h_{n+1}+h_n}.$$

Notice that as $r_n\to r$, i.e., $h_n\to0$, this s_n indeed converges to r, where the fraction is the error s_n-r. Recall

the assumption that $h_{n+1} = (c+\delta_n)h_n$ (which is obtained simply by rewriting $r_n - r$ as h_n). A bunch of brutal computations suggest that $\lim_{n\to\infty}(s_n - r)/h = 0$. □

Remark. The Aitken acceleration must be stopped once it gets stationary values, as subtractive cancellation in the formula will produce bad results eventually (recall we cannot do small number subtractions on machines).

Theorem 5.1.6

If λ is an eigenvalue of A and if A is nonsingular, then λ^{-1} is an eigenvalue of A^{-1}. *Proof: clear enough*:

$$Ax = \lambda x, x \neq 0 \implies x = A^{-1}Ax = A^{-1}\lambda x = \lambda(A^{-1}x) \implies A^{-1}x = \lambda^{-1}x.$$

Inverse Power Method

Now suppose we have A invertible with

$$|\lambda_1| \geqslant |\lambda_2| \geqslant \cdots \geqslant |\lambda_{n-1}| > |\lambda_n| > 0$$

(the opposite of our previous assumption), then we can invert A and use the power method on A^{-1} since

$$|\lambda_n^{-1}| > |\lambda_{n-1}^{-1}| \geqslant \cdots \geqslant |\lambda_1^{-1}| > 0.$$

However, we do not compute and solve $x^{(k+1)} = A^{-1}x^{(k)}$. Instead, we obtain $x^{(k+1)}$ by solving

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

which can be efficiently done using Gaussian elimination (factorization only needs to be done once since A is fixed).

Beginning of March 29, 2021

Other Variants

In addition to the power method and the inverse power method, we also have their **shifted** variants:
Suppose we are given A and $\Lambda(A) = \{\lambda_1, \ldots, \lambda_n\}$. Then $\Lambda(A - \mu I) = \{\lambda_1 - \mu, \ldots, \lambda_n - \mu\}$. The **shifter power method** uses the iterative method

$$x^{(k+1)} = (A - \mu I)x^{(k)}$$

(A replaced by $A - \mu I$). It becomes clear that this method computes the largest (absolute value) among $\{\lambda_1 - \mu, \ldots, \lambda_n - \mu\}$, i.e., the eigenvalue of A that is the farthest from μ.
By the same token, the **shifted inverse power method** uses

$$(A - \mu I)x^{(k+1)} = x^{(k)}$$

which (indirectly) computes the smallest among $\Lambda(A - \mu I)$, i.e., the eigenvalue of A closest to μ.

Since μ can be chosen arbitrarily, these provide a nice way to compute the "closest eigenvalues" to any $\mu \in \mathbb{R}$. This generalizes the power method in some sense. Suppose now we have a matrix with

$$|\lambda_1| = |\lambda_2| > |\lambda_3| > |\lambda_4| \geqslant |\lambda_5| \geqslant \cdots \geqslant |\lambda_n| > 0.$$

Clearly we can no longer use the (standard) power or inverse power methods (so they diverge). However, if we prescribe a μ close enough to $|\lambda_3|$, the shifted inverse power method will be able to give us $|\lambda_3|$ by iteration.

Now, the natural question is, how can we come up with a nice μ in the first place (i.e., how do we know the rough range of the eigenvalues), and how can we get more than one eigenvalues?

5.2 Schur's and Gershgorin's Theorems

Localizing Eigenvalues

Theorem 5.2.1: Gershgorin's Theorem

The spectrum (all eigenvalues) of an $n \times n$ matrix A is contained in the union of following n disks D_i in the complex plane, where each D_i is given by

$$D_i := \{z \in \mathbb{C} : |z - a_{i,i}| \leqslant \sum_{j \neq i} |a_{i,j}|\}.$$

(Compare this with diagonal dominance.) This *localizes* all the eigenvalues of A.

Remark. Once we know the regions in which the eigenvalues lie, we can use the shifted inverse method to begin "guessing" where exactly the eigenvalues are.

Proof. Let λ be any eigenvalue of A (so it's in the spectrum). Let x with $\|x\|_\infty = 1$ be the corresponding eigenvector, i.e., $Ax = \lambda x$. Assume the i^{th} component of x satisfies $x_i = 1$. Then

$$Ax = \lambda x \implies (Ax)_i = \sum_{j=1}^{n} a_{i,j} x_j = \lambda x_i.$$

Therefore

$$(\lambda - a_{i,i}) x_i = \sum_{j \neq i} a_{i,j} x_j.$$

Then taking absolute value on both sides gives

$$|(\lambda - a_{i,i}) x_i| = |\lambda - a_{i,i}| = \left| \sum_{j \neq i} a_{i,j} x_j \right| \leqslant \sum_{j \neq i} |a_{i,j}||x_j| \leqslant \sum_{j \neq i} |a_{i,j}|.$$

□

Now, a generalization of the Gershgorin's Theorem:

Theorem 5.2.2

If A is diagonalized diagonalized by $P^{-1}AP$ and if B is any matrix, then the eigenvalues of $A + B$ lie in the

union of disks D_i with

$$\{\lambda \in \mathbb{C} : |\lambda - \lambda_i| \leqslant \kappa_\infty(P)\|B\|_\infty\},$$

where $\lambda_1, \dots, \lambda_n$ are eigenvalues of A and $\kappa_\infty(P) = \|P\|_\infty \|P^{-1}\|_\infty$ (the conditional number w.r.t. $\|\cdot\|_\infty$).

Remark. If we take $A := \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ and $P = I$ and let B be any matrix with zeroes on diagonals, immediately we see that $A + B$ is just an "ordinary" matrix. Furthermore, $\kappa_\infty(P) = 1$ and $\|B\|_\infty$ is simply of form in Gershgorin's theorem (recall $\|B\|_\infty$ is the max of sum of absolute values of entries in one row, and B has zero diagonal).

Beginning of March 31, 2021

Proof. First recall that similar matrices have the same eigenvalues. Therefore the spectrum $\Lambda(A) = \Lambda(P^{-1}AP) =: \Lambda(D)$ where $D = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ (of A). In addition,

$$\Lambda(A + B) = \Lambda(P^{-1}(A + B)P) = \Lambda(D + P^{-1}BP) =: \Lambda(D + C).$$

If we apply Gershgorin's Theorem to $D + C$, then the eigenvalues of $A + B$ should lie in the Gershgorin disks of $D + C$, which are defined by

$$D_i = \left\{\lambda \in \mathbb{C} : |\lambda - (\lambda_i + c_{i,i})| \leqslant \sum_{j \neq i}^{n} |d_{i,j} + c_{i,j}| = \sum_{j \neq i}^{n} |c_{i,j}|\right\}.$$

Notice that

$$D_i \subset D_i' := \{\lambda \in \mathbb{C} : |\lambda - \lambda_i| \leqslant \sum_{j=1}^{n} |c_{i,j}|\}.$$

This follows from triangle inequality: if $\lambda \in D_i$,

$$|\lambda - \lambda_i - c_{i,i}| \leqslant \sum_{j \neq i}^{n} |c_{i,j}| \implies |\lambda - \lambda_i| \leqslant |\lambda - \lambda_i - c_{i,i}| + |c_{i,i}| \leqslant \sum_{j \neq i}^{n} |c_{i,j}| + |c_{i,i}| = \sum_{j=1}^{n} |c_{i,j}|.$$

But notice that the RHS is closely related to $\|C\|_\infty$: if we define

$$D_i'' := \{\lambda \in \mathbb{C} : |\lambda - \lambda_i| \leqslant \kappa_\infty(P)\|B\|_\infty\},$$

it follows that

$$\sum_{j=1}^{n} |c_{i,j}| \leqslant \max_{1 \leqslant i \leqslant n} \sum_{j=1}^{n} |c_{i,j}| = \|C\|_\infty = \|P^{-1}BP\|_\infty \leqslant \|P^{-1}\|_\infty \|B\|_\infty \|P\|_\infty = \kappa_\infty(P)\|B\|_\infty.$$

Therefore each $D_i \subset D_i' \subset D_i''$, and we are done! □

Schur's Factorization

Suppose we can find one eigenvalues. How can we find more? *Schur's factorization* gives us a way to "single out" the first eigenvalue from A.

Definition 5.2.3

A matrix U is **unitary** if $UU^* = I$ (where U^* denotes the conjugate transpose of U).

Theorem 5.2.4: Schur's Theorem

Every square matrix is unitarily similar to a triangular matrix, i.e., $A = U^{-1}BU$ for some unitary U. (In this case $A = U^{-1}BU = U^*BU$ as well.)

Proof. We prove this theorem by induction. Clear enough, the base case where A is 1×1 holds.
For the inductive step, suppose the theorem holds for all $(n-1)\times(n-1)$ matrices. Let $\lambda \in \Lambda(A)$ with x being the corresponding eigenvector, i.e., $Ax = \lambda x$. WLOG assume $\|x\|_2 = 1$. Now we define another scalar β depending simply on x_1 (first component) by $\beta = x_1/|x_1|$ if $x_1 \neq 0$ and 1 if $x_1 = 0$.
Claim that will be shown later (we will take this for granted for now): there exists a unitary U such that $Ux = \beta e^{(1)}$ (where $e^{(1)} := (1, 0, \dots)$). Then since $U^{-1} = U^*$ we have $b^{-1}x = U^*e^{(1)}$. Then

$$UAU^*e^{(1)} = UAb^{-1}x = \beta^{-1}\lambda Ux = \lambda e^{(1)}.$$

Notice that $[UAU^*]e^{(1)}$ is simply the first column of UAU^* by the construction of $e^{(1)}$. Once we know any $\lambda \in \Lambda(A)$, we are able to transform A into a matrix UAU^* whose first column is $(\lambda, 0, \dots)^T$. This is the first step towards arriving at a triangular matrix. Let $\tilde{A}$ be the bottom-right $(n-1)\times(n-1)$ submatrix of UAU^*. By induction hypnosis, there exist unitary $\tilde{U}$ such that $\tilde{U}\tilde{A}\tilde{U}^*$ is (upper) triangular. If we define

$$V = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix} U$$

it follows that V is unitary since $V^*V = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U}^* \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix} U = I$. Then

$$VAV^* = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix} UAU^* \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U}^* \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix}\begin{bmatrix} \lambda & * \\ 0 & \tilde{A} \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & \tilde{U}^* \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix}\begin{bmatrix} \lambda & *\tilde{U}^* \\ 0 & \tilde{A}\tilde{U}^* \end{bmatrix} = \begin{bmatrix} \lambda & *\tilde{U}^* \\ 0 & \tilde{U}\tilde{A}\tilde{U}^* \end{bmatrix}.$$

□

Corollary 5.2.5

Every square matrix is similar to a triangular matrix. *Trivial*.

Corollary 5.2.6

Every Hermitian matrix is unitarily similar to a diagonal matrix.

Proof. Let $A = A^*$. Since UAU^* is upper triangular for some unitary U, we have $(UAU^*)^*$ a lower triangular

matrix. However,

$$(UAU^*)^* = U^{**}A^*U^* = UAU^*$$

so UAU^* can only be diagonal. □

Now we show the lemma taken for granted in the proof of Schur's theorem.

Lemma 5.2.7

The matrix $I - vv^*$ is unitary if and only if $\|v\|_2^2 = 2$ or 0.

Proof. Notice that $(vv^*)^* = vv^*$. If $I - vv^*$ is unitary, then

$$\begin{aligned} I &= (I - vv^*)^*(I - vv^*) = (I - vv^*)(I - vv^*) \\ &= I - 2vv^* + vv^*vv^* \\ &= I - 2vv^* + (v^*v)(vv^*) \\ &= I - (2 - v^*v)vv^*, \end{aligned}$$

so if $I - vv^*$ is unitary, either $v^*v = 2$ or $vv^* = 0$. □

Lemma 5.2.8

Let x, y be two vectors such that $\|x\|_2 = \|y\|_2$ and $\langle x, y\rangle \in \mathbb{R}$. Then there exists a unitary matrix U of the form $I - vv^*$ such that $Ux = y$.

Proof. If $x = y$, simply let $v = 0$ so that $Ix = y$. If $x \neq y$, define

$$v = \frac{\sqrt{2}(x - y)}{\|x - y\|_2}.$$

Then

$$\begin{aligned} Ux - y &= (I - vv^*)x - y = x - vv^*x - y \\ &= x - y - \alpha^2(x - y)(x^* - y^*)x \\ &= (x - y)\left(1 - \alpha^2(x^*x - y^*x)\right). \end{aligned}$$

It turns out that $1 - \alpha^2(x^*x - y^*x) = 0$. Recall the assumption that $\|x\|_2 = \|y\|_2 \implies x^*x = y^*y$. Also, $\langle x, y\rangle \in \mathbb{R}$ so $\langle x, y\rangle = \langle y, x\rangle \implies x^*y = y^*x$. Then,

$$1 - \alpha^2(x^*x - y^*x) = 1 - \frac{\alpha^2}{2}(x^*x + y^*y - y^*x - x^*y) = 1 - \frac{\alpha^2}{2}(x^* - y^*)(x - y) = 1 - \frac{\alpha^2}{2}\|x - y\|_2^2 = 0.$$ □

Deflation

The creation of $\tilde{A}$ in Schur's theorem is called **deflation**. If an eigenvalue λ of A is known, we then have $\tilde{A}_{(n-1)\times(n-1)}$ with the same eigenvalues except λ (recall similar matrices have same eigenvalues). Then we can use power method again and find one particular eigenvalue of $\tilde{A}$ and then get a second eigenvalue of A. So on and so forth. The way to construct the corresponding U (for $\tilde{A}$) is given by obtaining v in the second lemma above and letting $U := I - vv^*$. To put formally, the steps are as follows:

(1) Obtain an eigenvector x corresponding to a known eigenvalue λ.

(2) Define $\beta := x_1/|x_1|$. If $x_1 = 0$ simply define $\beta = 1$.

(3) Define $\alpha = \sqrt{2}/\|x - \beta e^{(1)}\|_2$ and $v = \alpha(x - \beta e^{(1)})$.

(4) Let $U = I - vv^*$.

(5) Let $\tilde{A}$ be the $(n-1) \times (n-1)$ submatrix of UAU^*.

5.3 Orthogonal Factorization & Least-Square Problems

Let us recall that an inner product for vectors in $\mathbb{C}^n$ need to be non-degenerate, linear w.r.t. the first argument, and conjugate linear w.r.t. The second. Canonically, if $x = (x_1, \dots, x_n)^T$ and $y = (y_1, \dots, y_n)^T$ then

$$\langle x, y \rangle := \sum_{i=1}^{n} \overline{x_i} y_i = x^* y.$$

This also induces the norm $\|x\| = \sqrt{\langle x, x \rangle}$.

Beginning of April 5, 2021

(*Actually, the inner product can also be defined in other ways: for example define $\langle x, y \rangle := x^* A y$ where A is positive definite*.)

Definition 5.3.1

A set of vectors $\{v_1, \dots, v_n\}$ is said to be **orthogonal** if $\langle v_i, v_j \rangle = 0$ whenever $i \neq j$. They are said to be **orthonormal** if $\langle v_i, v_j \rangle = \delta_{i,j}$ (the Kronecker delta).

Proposition 5.3.2

The **Pythagorean rule** remains valid in inner product space: if $\langle x, y \rangle = 0$ then

$$\|x + y\|^2 = \|x\|^2 + \|y\|^2$$

and in particular we are interested in $\|\cdot\|_2$. Indeed,

$$\|x + y\|^2 = \langle x + y, x + y \rangle = \langle x, x \rangle + \langle y, y \rangle + 2\mathfrak{Re}\,\langle x, y \rangle = \|x\|^2 + \|y\|^2.$$

Gram-Schmidt Orthogonalization

Suppose we have a set of linearly independent vectors $\{x_1, x_2, \dots\}$ and we want to obtain a set of orthonormal vectors $\{u_1, u_2, \dots\}$. To begin, we begin by normalizing v_1 and set $u_1 := v_1/\|v_1\|$. Then inductively we define

$$u_k' = x_k - \sum_{i<k} \langle x_k, u_i \rangle u_i$$

and set $u_k := u'_k/\|u'_k\|$. This is called the **Gram-Schmidt process**. The summation $\sum_{i<k} \langle x_k, u_i\rangle u_i$ is the projection of x onto the subspace spanned by $\{u_1, \dots, u_{k-1}\}$ which is equivalent to that of $\{x_1, \dots, x_{k-1}\}$. After all, each u_i is a linear combination of $x_1, \dots, x_i$.

Theorem 5.3.3

The finite-truncation $\{u_1, u_2, \dots, u_n\}$ of the Gram-Schmidt sequence is an orthonormal basis for the linear span of $\{x_1, x_2, \dots, x_n\}$.

Proof. We proceed by induction; the base case is clearly true. Now assume the case for $k-1$ holds. Define

$$v = x_k - \sum_{i<k} \langle x_k, u_i\rangle u_i.$$

Then, for $j < k$,

$$\langle v, u_j\rangle = \langle x_k, u_j\rangle - \sum_{i<k} \langle x_k, u_i\rangle \langle u_i, u_j\rangle = \langle x_k, u_j\rangle - \sum_{i<k} \delta_{i,j} \langle x_k, u_i\rangle = \langle x_k, u_j\rangle - \langle x_k, u_j\rangle = 0.$$

Note that $v \neq 0$ or otherwise x_k is in the span of $\{u_1, \dots, u_{k-1}\}$ which by hypothesis is also that of $\{x_1, \dots, x_{k-1}\}$, contradicting the linear independence of $\{x_n\}$. Hence $v \neq 0$ and $u_k := v/\|v\|$ form an orthonormal basis along with the previous $k-1$ u_i's. The claim then follows as clearly $\text{span}(u_1, \dots, u_k) \subset \text{span}(x_1, \dots, x_k)$ and they have to equal as their dimensions agree. □

Theorem 5.3.4

If we apply Gram-Schmidt to an $m \times n$ matrix A of rank n then we obtain the factorization $A = BT$ where $B_{m\times n}$ has orthonormal columns and T is upper triangular with positive diagonal.

Proof. Heuristically we start with the n linearly independent column vectors of A (since it has rank n) and Gram-Schmidt gives a bunch of orthonormal vectors which goes into B. Then T records all the coefficients of the linear combinations and the diagonal terms are the so-called "scaling factor" $1/\|u'_k\|$ obtained in $u'_k/\|u'_k\|$. To put formally,

$$x_k = \sum_{i<k} c_{i,k} u_i \implies \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ & c_{2,2} & \cdots & c_{2,n} \\ & & \ddots & \vdots \\ & & & c_{n,n} \end{bmatrix}.$$ □

Remark. If we use the standard inner product $\langle x, y\rangle := x^* y$ and $m = n$ then B is unitary.

Modified Gram-Schmidt

In the modified version, we define

$$v_k := x_k - \sum_{i<k} \frac{\langle x_k, v_i\rangle v_i}{\langle v_i, v_i\rangle}.$$

It follows that we no longer need to manually normalize v_k in this way.

Least-Squares Problem

This is an important application of orthogonal factorization. Least-squares problems are widely examined in topics like regression analysis (maximum likelihood estimate, MLE), data fitting, optimization, and so on.

Consider a system of m equations and n unknowns written as $Ax = b$ where A is of $m \times n$, $x \in \mathbb{R}^n$ (or $\mathbb{C}^n$), and $b \in \mathbb{R}^m$ (or $\mathbb{C}^m$). We assume the rank of A is n. It immediately follows that $m \geqslant n$. Therefore there are more (or same) equations than unknowns. If $m > n$, often times the system has no precise solutions. The most natural way that follows is to find a "best approximation", a vector x such that $\|b - Ax\|$ attains its minimum. We use $\|\cdot\|_2$ (because of some statistical reasons due to Gauß).

Lemma 5.3.5

If x satisfies $A^*(Ax - b) = 0$ then x solves the least-squares problem.

Beginning of April 9, 2021

Proof. Suppose x solves $A^*(Ax-b) = 0$ and let y be any other point. By assumption $Ax-b$ (or $b-Ax$) is orthogonal to the column space of A. On the other hand, $A(x - y)$ is in the column space of A, so $\langle b - Ax, A(x - y)\rangle = 0$. Then, by Pythagorean rule (and orthogonality)

$$\|b - Ay\|_2^2 = \|b - Ax + A(x - y)\|_2^2 = \|b - Ax\|_2^2 + \|A(x - y)\|_2^2 \geqslant \|b - Ax\|_2^2. \qquad \square$$

Remark. If $\operatorname{rank}(A_{m\times n}) = n$ then so is A^*A, an $n \times n$ matrix, so it's nonsingular. Then if for some x we have $A^*(Ax - b) = 0$ it must be unique. The lemma, combined with the rank assumption, gives us a unique least-squares solution.

Now the question becomes how to find the least-squares solution?
If A has been factorized as $A = BT$ ($A_{m\times n}$, $B_{m\times n}$ with orthonormal columns, and $T_{n\times n}$ upper triangular with positive diagonal), then the least-squares solution is the one that solves

$$Tx = (B^*B)^{-1}B^*b$$

(indeed this is just the one for $A^*Ax = A^*b$ which can be re-written as $T^*B^*BTx = T^*B^*B(B^*B)^{-1}B^*b = T^*B^*b$). Alternatively, we can solve the normal equation $A^*Ax = A^*b$, but this needs the additional assumption that rank n so that A^*A is nonsingular, Hermitian, and positive definite. Hence we can invoke Cholesky factorization.

However, we often choose the first approach over the second as it sometimes provides a better condition number:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix} \Longrightarrow A^*A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} + \epsilon^2 I_{3\times 3}.$$

Heuristically, ϵ prevents A from having rank 1 but only ϵ^2 prevents A^*A from having rank 1. For smaller ϵ's we may run into issues, e.g., when ϵ is significant but ϵ^2 too small such that it is ignored by the computer.

Householder's QR Factorization

Goal: for a $m \times n$ matrix A, we want to factorize it into $A = QR$ where Q is $m \times m$ unitary and R $m \times n$ upper triangular. (Compare this to $A = BT$ in Gram-Schmidt; the main difference is that here Q is square whereas in Gram-Schmidt T is square. If A is square then the two methods are equivalent.)

5.4 Singular-Value Decomposition & Pseudoinverses

Now we generalize diagonalization $A = Q\Lambda Q^{-1}$ for real symmetric matrices and Schur's factorization $A = U^*BU$ (U unitary and B upper triangular) for a square matrix to an arbitrary matrix. The process is called **singular value decomposition** (SVD).

Theorem 5.4.1

Any arbitrary complex $m \times n$ matrix A can be factorized as

$$A = PDQ$$

where P is $m \times m$ unitary, D $m \times n$ diagonal, and Q $n \times n$ unitary.

Proof. Notice that A^*A is $n \times n$, Hermitian, and positive semidefinite:

$$(A^*A)^* = A^*A^{**} = A^*A \text{ and } x^*(A^*Ax) = (Ax)^*(Ax) \geqslant 0.$$

Beginning of April 12, 2021

Therefore the eigenvalues of A^*A is real and nonnegative, and it is well-defined to write them as $\sigma_1^2, \dots, \sigma_n^2$ in descending order. Then $\sigma_1, \dots, \sigma_n$ are called the **singular values** of A. Suppose $\sigma_1 > \cdots > \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$. Let $\{u_1, \dots, u_n\}$ be an orthonormal set of eigenvectors corresponding to $\{\sigma_1^2, \dots, \sigma_n^2\}$. (This is possible because a Hermitian matrix can be diagonalized.) Then since

$$\|Au_i\|_2^2 = u_i^*A^*Au_i = u_i^*\sigma^2u_i = \sigma^2u_i^*u_i = \sigma^2,$$

we see that $Au_i = 0$ for $i \geqslant r+1$. Also, notice that $r = \text{rank}(A^*A) \leqslant \min(\text{rank}(A^*), \text{rank}(A)) \leqslant \min(m, n)$, which will be useful when we construct P and Q.

We first construct $Q_{n\times n}$ by setting its rows to be $u_1^*, \dots, u_n^*$. Clearly Q is unitary as the u_i's are orthonormal.

Now we define $v_i := \sigma^{-1}Au_i$ for $1 \leqslant i \leqslant r$. Clearly v_i's are also orthonormal:

$$v_i^*v_j = \sigma_i^{-1}(Au_i)^*\sigma_j^{-1}(Au_j) = \frac{u_i^*A^*Au_j}{\sigma_i\sigma_j} = \frac{\sigma_j^2u_i^*u_j}{\sigma_i\sigma_j} = \delta_{i,j}.$$

However, $\{v_1, \dots, v_r\}$ might not be enough for our $P_{m\times m}$, and we might need more orthonormal vectors. We can easily overcome this by extending $\{v_1, \dots, v_r\}$ to $\{v_1, \dots, v_r, \dots, v_m\}$, an orthonormal basis for $\mathbb{C}^m$. In fact,

this choice can be arbitrary (as the bottom part of D's diagonal are zeros, i.e., these arbitrarily extended vectors correspond to the zero singular value which makes no difference at all).

We now claim

$$A = PDQ = \begin{bmatrix} | & \cdots & | \\ v_1 & \cdots & v_m \\ | & \cdots & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \begin{bmatrix} — & u_1^* & — \\ \vdots & \vdots & \vdots \\ — & u_n^* & — \end{bmatrix}$$

or equivalently $D = P^* A Q^*$. Indeed, for $\leqslant i \leqslant r$,

$$(P^* A Q^*)_{i,j} = v^* A u_j = (\sigma^{-1} A u_i)^* A u_j = \sigma v_i^* A^* A u_j = \sigma_i \delta_{i,j},$$

and if $m \geqslant i \geqslant r+1$, $(P^* A Q^*)_{i,j} = 0$ as v_{r+1} is orthogonal to span$(u_1, \ldots, u_r)$. □

Pseudoinverse

As suggested by the name, a pseudoinverse is a generalization of an inverse.

Given an $m \times n$ "diagonal" matrix D with "diagonal" entries $\sigma_1, \ldots, \sigma_r, 0, \ldots$ in which $\sigma_i > 0$, we define the **pseudoinverse** D^+ to be the $n \times m$ "diagonal" matrix with "diagonal" entries $\sigma_1^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots$. More generallly, if $A = PDQ$ then

$$A^+ = Q^* D^+ P^*.$$

For an invertible matrix, $A^{-1}A = I$, but for pseudoinverses, $A^+ A$ usually is of form $\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix}$ (where 0 denotes a block of appropriate size). Nevertheless we have $AA^+A = A$, $A^+AA = A$, $(AA^+)^* = AA^+$, and $(A^+A)^* = A^+A$. Notice that the SVD is not unique but the pseudoinverse is uniquely determined. We will show this later.

Beginning of April 14, 2021

Inconsistent and Underdetermined Systems

A system is **consistent** if and only if there exists a solution.

Consider $Ax = b$ where A is $m \times n$, x is $n \times 1$, and b is $m \times 1$. We have the following scenarios, each of which defines the corresponding **minimal solution** to the system:

(1) The system is consistent and has a unique solution x. If so, x is the minimal solution.

(2) The system is consistent and has a set of solutions. The minimal solution is defined to be the one with the least Euclidean norm (infimum, to be exact).

(3) If the system is inconsistent and there exists a unique least-squares solution x (e.g., when rank$(A) = n$ so A^*A is nonsingular), then the minimal solution is simply x.

(4) If the system is inconsistent and has a set of least-squares solutions, the minimal solution is the one with the least Euclidean-norm (infimum, to be exact).

To put into formal mathematical language, if $\rho := \inf\{\|Ax - b\|_2 : x \in \mathbb{C}^n\}$ then the minimal solution of $Ax = b$ is the element with the least norm in the set $\{x : \|Ax - b\|_2 = \rho\}$. In (1) and (3) we have $\rho = 1$; in (2) and (4), $\rho > 0$.

Remark. The infimum can indeed be obtained by some x because $Ax - b$ is linear. Furthermore, it can be characterized by the following theorem.

Theorem 5.4.2

The minimum solution of the equation $Ax = b$ is given by the pseudoinverse $x = A^+b$.

Proof. Let $A = PDQ$ be the SVD of A. Since Q is unitary, it is in particular nonsignular and thus $Q : \mathbb{C}^n \to \mathbb{C}^n$ is surjective. Therefore

$$\begin{aligned}
\rho &= \inf_{x\in\mathbb{C}^n} \|Ax - b\|_2 = \inf_x \|PDQx - b\|_2 \\
&= \inf_x \|P^*(PDPx - b)\|_2 && (P^* \text{ unitary and } \|P^*v\| = \|v\|) \\
&= \inf_x \|DQx - P^*b\| = \inf_y \|Dy - c\|. && (\text{where } y := Qx \ \& \ c := P^*b)
\end{aligned}$$

Since D is $\operatorname{diag}(\sigma_1, \dots, \sigma_r, 0, \dots)$, we know

$$\begin{aligned}
\|Dy - c\|_2^2 &= \text{norm}^2 \text{ of } \begin{bmatrix} \lambda_1 y_1 - c_1 & \cdots & \lambda_r y_r - c_r & -c_{r+1} & \cdots & -c_m \end{bmatrix}^T \\
&= \sum_{i=1}^{r} |\lambda_i y_i - c_i|^2 + \sum_{i=r+1}^{m} |c_i|^2.
\end{aligned}$$

Clearly this quantity is minimized by letting $y_i = c_i/\sigma_i$ for each $i \in [1, r]$. (Note that we have no control over the remaining c_i's as those are determined by A. Therefore all we can do is to set the first sum to 0, which we can.) Then,

$$\rho = \left(\sum_{i=r+1}^{m} c_i^2 \right)^{1/2}$$

and since this ρ does not depend on values of $y_{r+1}, \dots, y_n$, we want to set them to all 0 to minimize $\|y\|_2$. In other words, the minimal solution is

$$y = \begin{bmatrix} c_1/\sigma_1 & \cdots & c_r/\sigma_r & 0 & \cdots \end{bmatrix}^T.$$

It is easy to see that this is in fact given by $y = D^+c$ (recall the nonzero diagonal entries of D^+ is simply $1/\sigma_i$), and indeed since y is defined to by Qx,

$$x = Q^{-1}y = Q^*y = Q^*D^+c = Q^*D^+P^*b = A^+b. \qquad \square$$

Remark. In MATLAB we can simply use `x = A \ b` to compute the minimal solution.

Now we prove rigorously that pseudoinverses (and minimal solutions) are unique.

Theorem 5.4.3: Penrose properties

(R. Penrose, 1955) For any matrix A, there exists at most one matrix X (and also at least one, namely A^+,

which we will show later) having these four properties:

$$(1)AXA = A \quad (2)XAX = X \quad (3)(AX)^* = AX \quad (4)(XA)^* = XA$$

Beginning of April 16, 2021

Proof. Suppose X and Y both satisfy all the properties above. Then

$$\begin{aligned}
X &= XAX = X(AYA)X && (2)\&(1)\\
&= X(AYA)Y(AYA)X && (1)\\
&= (XA)^*(YA)^*Y(AY)^*(AX)^* && (3)\&(4)\\
&= (A^*X^*A^*)Y^*YY^*(A^*X^*A^*)\\
&= (AXA)^*Y^*YY^*(AXA)^*\\
&= A^*Y^*YY^*A^* = (YA)^*Y(AY)^* && (1)\\
&= YAYAY = YAY = Y. && (3)\&(4),\ \text{then } (2)
\end{aligned}$$

□

Remark. This proof is very similar to showing how A^{-1} is unique, should it exist: if X, Y are inverses of A then $X = XI = XAY = IY = Y$. Except here it requires more steps.

Theorem 5.4.4

Connecting to the previous theorem, the pseudoinverse of a matrix has those four Penrose properties. Hence each matrix has a unique pseudoinverse.

Proof. Let $A = PDQ$ be some SVD. Then $A^+ = Q^*D^+P^*$. We will show that A^+ satisfies all four properties as listed above. First off:

$$AA^+A = (PDQ)(Q^*D^+P^*)(PDQ) = P(DD^+D)(Q) = DD^+D = D.$$

Hence (1) holds ($DD^+D = D$ is clear; they are all diagonal matrices). Likewisae,

$$A^+AA^+ = \cdots = Q^*(D^+DD^+)P^* = Q^*D^+P^* = A^+$$

and (2) holds. (3) and (4) are similar and are omitted. □

Some more remarks on SVD: suppose $A = PDQ$ and $D = \text{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots)$. Then

(1) $\text{rank}(A) = r$: *since* $\text{rank}(XY) \leqslant \min(\text{rank}(A), \text{rank}(B))$*, we have*

$$\text{rank}(PDQ) \leqslant \text{rank}(A) \implies \text{rank}(A) \leqslant \text{rank}(D)$$

and

$$P^*AQ^* = D \implies \text{rank}(A) \geqslant \text{rank}(D)$$

(2) $\{v_1, \ldots, v_r\}$ is an orthonormal basis for the range of A

(3) $\{u_{r+1}, \ldots, u_n\}$ is an orthonormal basis for the null space of A:.

(4) $\|A\|_2 = \max|\sigma_i|$. WLOG if we let σ_1 be the largest singular value then

$$\|A\|_2 = \sup_{\|x\|_2=1} \|Ax\|_2 = \sup_{\|x\|_2=1} \|PDQx\|_2 = \sup_{\|x\|_2=1} \|DQx\|_2$$

$$[y \coloneqq Qx;\ Q \text{ unitary}] = \sup_{\|y\|_2=1} \|Dy\|_2 = \|De^{(1)}\| = \sqrt{\sigma_1^2} = |\sigma_1|.$$

Chapter 6

Approximating Functions

6.1 Polynomial Interpolation

In computer, function values are stored discretely (e.g., for $f : (a, b) \to \mathbb{R}$, the computer clearly cannot store the values of all $f(x)$ for $x \in (a, b)$). (We are talking about the classical methods, not symbolic functions or things like that.)

Question: given a set of data about the function in the format below, how do we recover the original function?

x	x_0	x_1	x_2	$\dots$	x_n
$y := f(x)$	y_0	y_1	y_2	$\dots$	y_n

There are usually two ways: **interpolation** and **approximation** (least-squares approximation in $\|\cdot\|_2$ and Chebyshev best approximation in $\|\cdot\|_1$). We will look at the former first.

Beginning of April 19, 2021

Given the above table containing $(x_0, y_0), ..., (x_n, y_n)$, we seek a polynomial p of lowest possible degree for which $p(x_i) = y_i$ for all i. Such p is said to **interpolate** the data.

Theorem 6.1.1

If $x_0, ..., x_n$ are distinct real numbers, then for any $y_0, ..., y_n$ there exists a unique polynomial p_n of degree at most n satisfying $p_n(y_i) = y_i$ for $1 \leqslant i \leqslant n$.

Proof. We first show uniqueness: suppose p_n and q_n are two polynomials of equal degree, both $\leqslant n$. Suppose $(p_n - q_n)(x_i) = 0$ for $1 \leqslant i \leqslant n$. If $p_n - q_n \not\equiv 0$ then it is a polynomial with $n+1$ zeros, so it must be of order at least $n+1$. Contradiction. Hence $p_n \equiv q_n$.

Now we show existence. The proof can be done by *Newton's form*, *Lagrange's form*, or the *Vandermonde matrix*. Here we use the first, and we proceed by induction. The base case is clearly true: for a degree 0 polynomial and (x_0, y_0), simply let $p_0(x_0) = y_0$ the constant function. Now assume that for $(x_0, y_0), ..., (x_{k-1}, y_{k-1})$ we can find some polynomial p_{k-1} satisfying the assumptions. Now suppose we are given an additional (x_k, y_k). Consider

$$p_k(x) = p_{k-1}(x) + c(x - x_0)(x - x_1)...(x - x_{k-1}).$$

It follows that p_k and p_{k-1} agree at $x_0, ..., x_{k-1}$ as the second term evaluates to 0. It is also clear that $p_k(x)$ is a polynomial of degree not exceeding k. Now we just need to find c, and this is subject to the condition $p_k(x_k) = y_k$, i.e.,

$$c = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1)...(x_k - x_{k-1})}.$$

Since the x_i's are distinct, the denominator is nonzero and this division is well-defined. The claim follows. □

Remark. There is no remark that $x_0 < x_1 < ... < x_n$; that is, the method stays unaffected if we shuffle the order of (x_i, y_i)'s so it generates the same p_n. This is directly guaranteed by the uniqueness part.

Beginning of April 21, 2021

Newton Form

What we have been basically doing above can be summarized by

$$p_n(x) = \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1} (x - x_j).$$

Lagrange Form

Here we consider the same set of data $((x_0, y_0), ..., (x_n, y_n))$. Instead of expressing $p_n(x)$ as a sum of polynomials of (possibly) increasing degrees all the way till degree n, we consider

$$p_n(x) = y_0\ell_0(x) + ... + y_n\ell_n(x) = \sum_{i=0}^{n} y_i\ell_i(x),$$

such that

$$\ell_0(x_0) = 1, \ell_0(x_1) = 0, ..., \text{ and in general } \ell_i(x_j) = \delta_{i,j}.$$

We say that the polynomials $\ell_0, ..., \ell_n$ are polynomials that depend on the **nodes** $x_0, ..., x_n$ but <u>not</u> on the **ordinates** $y_0, ..., y_n$. It immediately follows that

$$p_n(x_i) = \sum_{j=0}^{n} y_j\delta_{i,j} = y_i\delta_{i,i} = y_i, \text{ as desired.}$$

So, how to compute the polynomials? Take ℓ_0 for example. Clearly, to be 0 at $x_1, ..., x_n$, it needs to be of form

$$\ell_0(x) = c(x - x_1)...(x - x_n) = c\prod_{i=1}^{n}(x - x_i).$$

What is c? It is obtained by substituting $x = x_0$ and solving $\ell_0(x_0) = 1$, namely

$$c = 1/\prod_{i=1}^{n}(x_0 - x_i) = \prod_{i=1}^{n}(x_0 - x_i)^{-1}.$$

To generalize this,

$$\ell_i(x) = \left[\prod_{j\neq i}(x_i - x_j)^{-1}\right]\prod_{j\neq i}(x - x_j) = \prod_{j\neq i}\frac{x - x_j}{x_i - x_j}.$$

These functions ℓ are called the **cardinal functions**, and with them, we obtain the **Lagrange form** of the interpolating polynomials.

Remark. This is more intuitive and is often used in proofs. For computers, however, this is a heavy task.

The Vandermonde Matrix

Suppose our polynomial of interest looks like

$$p_n(x) = a_0 + a_1x + ...a + nx^n = \sum_{i=0}^{n} a_i x^i,$$

where we are given the same conditions ($n+1$ pairs of x and y). How to find $p_n(x)$? LINEAR ALGEBRA!!

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

The humongous matrix on the left is called the **Vandermond matrix**. If the x_i's are distinct then it is nonsingular (indeed we have a trivial nullspace). In fact this number is given by

$$\prod_{0 \leqslant i < j \leqslant n} (x_i - x_j).$$

It is often times ill conditioned: if one of the x_i's has absolute value > 1 then x_i^n may become huge. Recall how ϵ's may spoil a matrix. The conditional number $\kappa(A)$ for such a matrix is huge.

Comparison

Now we have the following forms:

(1) Newton: $p_n(x) = c_0 + c_1(x - x_0) + ... + c_n(x - x_0)...(x - x_{n-1})$.

(2) Lagrange: $p_n(x) = d_0\ell_0(x) + d_1\ell_1(x) + ... + d_n\ell_n(x)$.

(3) Vandermonde: $p_n(x) = a_0 + a_1x + ... + a_nx^n$.

Notice that $\{1, (x - x_0), ..., (x - x_0)...(x - x_{n-1})\}$, $\{\ell_0(x), ..., \ell_n(x)\}$, and $\{1, x, ..., x^n\}$ are three bases of the linear space containing all polynomials of degree $\leqslant n$.

Newton form:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & 0 & \cdots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \cdots & \prod_{i=0}^{n-1}(x_n - x_i) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

We need to solve a system involving a triangular system. Fine.

Lagrange form: this is even nicer.

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Some More Remarks

(1) Since the Vandermonde matrix is often ill conditioned, we usually avoid using it in the practical world.

(2) Newton form is usually a better numerical method to solve the polynomial interpolation for the following reasons:

(a) It is an inductive algorithm so if more date points are added later on, we don't need to start from scratch again. Instead we can use what we already have; in other words Newton form can be easily extended.

(b) It can be combined with the *divided difference algorithm* (to be discussed).

Beginning of April 23, 2021

Error in Polynomial Interpolation

Theorem 6.1.2

Let f be a function in $C^{n+1}[a,b]$. Let p be the polynomial of degree $\leqslant n$ that interpolates f at $n+1$ points $(x_0,\ldots,x_n)$ in the interval. Then, to each $x \in [a,b]$ there exists a $\xi_x \in [a,b]$ such that

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

Proof. The term $f^{(n+1)}(\xi_x)$ is a clear hint for Taylor expansion. First notice that if x is one of the nodes (i.e., $x = x_i$) then the claim is trivial. Now suppose x is not a node. Define functions

$$\omega(t) = \prod_{i=0}^{n} (t - x_i) \qquad \varphi(x) = f(x) - p(x) - \lambda\omega(x)$$

where $\lambda \in \mathbb{R}$ makes $\varphi(x) = 0$. In other words,

$$\lambda = \frac{f(x) - p(x)}{\omega(x)}.$$

Our goal is to show that λ can be represented by $\frac{1}{(n+1)!} f^{(n+1)}(\xi_x)$. Notice that φ not only vanishes at x but also $x_0,\ldots,x_n$. Also notice that $\varphi \in C^{n+1}[a,b]$. By Rolle's theorem, φ' has at least $n+1$ distinct zeros in (a,b), and

inductively $\varphi^{(n+1)}$ has at least one zero, say $\xi_x \in (a,b)$. This means

$$\begin{aligned}
0 = \varphi^{n+1}(\xi_x) &= f^{(n+1)}(\xi_x) - p^{(n+1)}(\xi_x) - \lambda\omega^{(n+1)}(\xi_x) \\
&= f^{(n+1)}(\xi_x) - 0 - \lambda\omega^{(n+1)}(\xi_x) && (\deg(p) \leqslant n) \\
&= f^{(n+1)}(\xi_x) - \lambda \cdot \frac{\mathrm{d}^{n+1}}{\mathrm{d}t^{n+1}}\Big[\prod_{i=0}^{n}(t-x_i)\Big]_{t=\xi_x} \\
&= f^{(n+1)}(\xi_x) - \lambda \cdot \frac{\mathrm{d}^{n+1}}{\mathrm{d}t^{n+1}}\left[t^{n+1} + c_n t^n + \ldots\right]_{t=\xi_x} \\
&= f^{(n+1)}(\xi_x) - (n+1)!\lambda = f^{(n+1)}(\xi_x) - (n+1)!\frac{f(x)-p(x)}{\omega(x)},
\end{aligned}$$

and the claim follows. □

Remark. In the above expression, the only term that we have little control of is $f^{(n+1)}(\xi_x)$. Everything else can be easily bounded. If we can manage to bound $f^{(n+1)}(\xi_x)$ then we can bound the entire error.

Example 6.1.3. Consider $f(x) = \sin x$ and we approximate it by a polynomial of degree 9 with 10 notes on $[0,1]$. Clearly $f^{(10)}(\xi_x)$ is just a trig function whose absolute value is bounded by 1. It is also clear that $\prod_{i=0}^{9}|x-x_i| \leqslant 1$ (of course we can further bound it). Then

$$|\sin x - p(x)| \leqslant \frac{1}{10!} < 2.8 \cdot 10^{-7}.$$

We clearly see that if we interpolate the sine function using more nodes, then the error can be greatly reduced — it decays factorially.

Chebyshev Polynomials

The optimization of the above error leads to a system of polynomials of polynomials called the **Chebyshev polynomials**. As mentioned above, there is a way to minimize the term $\prod_{i=0}^{n}(x-x_i)$. They are defined iteratively on $[0,1]$ (we've mentioned this once), and if the domain changes, we need to translate and scale accordingly:

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

Beginning of April 26, 2021

Theorem 6.1.4

For $x \in [-1,1]$, the Chebyshev polynomials have the closed-form formula

$$T_n(x) = \cos(n\cos^{-1}(x)).$$

Proof. Indeed, $T_0(x) = \cos(0 \cdot \cos^{-1}(x)) = 1$ and $T_1(x) = \cos(1 \cdot \cos^{-1}(x)) = x$. Now notice that

$$\cos(A+B) = \cos A \cos B - \sin A \sin B,$$

so

$$\cos(n+1)\theta = \cos\theta\cos(n\theta) - \sin\theta\sin(n\theta)$$
$$\cos(n-1)\theta = \cos(\theta\cos(n\theta) + \sin\theta\sin(n\theta)$$

so

$$\cos(n+1)\theta = 2\cos\theta\cos(n\theta) - \cos(n-1)\theta.$$

Substituting $\theta := \cos^{-1} x$ and $x = \cos\theta$, we have

$$\begin{cases} f_0(x) = 1 \qquad f_1(x) = x \\ f_{n+1}(x) = 2xf_n(x) - f_{n-1}(x) \end{cases} \quad \text{where } f_n(x) = \cos(n\cos^{-1}(x)).$$ □

From this theorem we immediately obtain the following properties:

$$|T_n(x)| \leqslant 1 \qquad x \in [-1,1]$$
$$T_n(\cos(k\pi/n)) = \cos(k\pi) = (-1)^k \qquad 0 \leqslant k \leqslant n$$
$$T_n(\cos[(2k-1)/(2k)]\pi) = 0 \qquad 1 \leqslant k \leqslant n$$

Note that T_n is a degree-n polynomial with leading term $2^{n-1}x^n$. Therefore $2^{1-n}T_n$ is a **monic** polynomial.

Theorem 6.1.5

If p is a monic polynomial of degree n, then

$$\|p\|_\infty = \max_{-1\leqslant x\leqslant 1} |p(x)| \geqslant 2^{1-n} = \|2^{1-n}T_n\|_\infty.$$

This bounds provides an estimate for $\prod_{i=0}^{n}(x - x_i)$.

Proof. Suppose the contradiction that $|p(x)| < 2^{1-n}$ for all $x \in [-1,1]$. Let $q := 2^{1-n}T_n$ and $x_k = \cos(k\pi/n)$. Since q is a monic polynomial of degree n, we have

$$(-1)^k p(x_k) \leqslant |p(x_k)| \leqslant 2^{1-n} = (-1)^k q(x_k).$$

Therefore

$$(-1)^k[q(x_k) - p(x_k)] > 0,$$

i.e., $q(x_k) - p(x_k)$ oscillates between positive and negative values. It follows that $q - p$ a continuous polynomial must have at least n roots on $[-1,1]$, contradicting the fact that both are monic which implies $\deg(p-q) \leqslant n-1$. Therefore such p cannot exist. □

Choosing the Nodes

It follows from

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

that

$$\sup_{|x|\leqslant 1}|f(x) - p(x)| \leqslant \frac{1}{(n+1)!}\sup_{|x|\leqslant 1}|f^{(n+1)}(x)| \sup_{|x|\leqslant 1}\left|\prod_{i=0}^{n}(x - x_i)\right|,$$

since all the ξ_x's must also lie within the same interval. From above we see that the last term is at least 2^{-n} (note we have $n+1$ products, not n, hence 2^{-n} not 2^{1-n}). Thus, the minimum is attained if the last term is the "normalized" Chebyshev polynomial $2^{-n}T_{n+1}$. If so, it becomes clear that the notes are

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

Beginning of April 29, 2021

6.2 Divided Differences

In this section we develop a specific algorithm to obtain the coefficients for the Newton form.
To start simple, consider $p_2(x) = c_0q_0(x) + c_1q_1(x) + c_2q_2(x)$ where the given data are $(x_0, y_0), (x_1, y_1)$, and (x_2, y_2). By definition

$$q_0(x) = 1 \quad q_1(x) = (x - x_0) \quad q_2(x) = (x - x_0)(x - x_1).$$

(1) We begin by solving $p_0(x) = c_0q_0(x)$, a solution that interpolates (x_0, y_0). Clearly $q_0 = y_0 =: f[x_0]$.

(2) Now we solve $p_1(x) = c_0q_0(x) + c_1q_1(x) = y_0 + c_1(x - x_0)(x - x_1)$, a polynomial that interpolates (x_0, y_0) and (x_1, y_1). Clearly $p_1(x)$ interpolates (x_0, y_0) regardless of value of c_1, so we need to make sure p_1 interpolates (x_1y_1). This means

$$p_1(x_1) = y_0 + c_1(x_1 - x_0) = y_1 \implies c_1 = \frac{y_1 - y_0}{x_1 - x_0} =: f[x_0, x_1].$$

(3) Now we consider the last term c_2 and p_2. Similarly, we only need to focus on the term containing c_2 because (x_0, y_0) and (x_1, y_1) are automatically interpolated. Then

$$\begin{aligned} &p_2(x_2) = f[x_0] + f[x_0, x_1](x - x_0) + c_2(x_2 - x_0)(x_2 - x_1) = y_2 \\ \implies &c_2 = \frac{y_2 - f[x_0] - f[x_0, x_1](x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{(y_2 - y_0)/(x_2 - x_0) - f[x_0, x_1]}{x_2 - x_1} = \frac{f[x_0, x_2] - f[x_0, x_1]}{x_2 - x_1} =: f[x_0, x_1, x_2]. \end{aligned}$$

We adopt the notation $f[x_0, ..., x_n]$ to stress the fact that the coefficient depends only on the x_i's included in the bracket. To sum it up,

$$p_2(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$$

Definition 6.2.1

In Newton form, if $p_n(x) = \sum_{i=0}^{n} c_iq_i(x)$ where $q_i(x) = \prod_{j=0}^{i-1}(x - x_j)$, then we define

$$c_i := f[x_0, \ldots, x_i].$$

These $f[x_0, \ldots, x_i]$ are called the **divided differences** of f.

Proposition 6.2.2

The divided difference is a symmetric function of its arguments, i.e., if $\{z_0, z_1, ..., z_n\} = \{x_0, x_1, ..., x_n\}$ then

$$f[z_0, z_1, ..., z_n] = f[x_0, x_1, ..., x_n].$$

Proof. Recall that the polynomial p_n that interpolates the data is unique. Both the LHS and RHS denotes the coefficient of the term x^n and they are of course the same. □

Theorem 6.2.3

This theorem provides a recursive formula for the divided difference:

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

Remark. With this theorem, when we are given $(x_0, f(x_0)), (x_1, f(x_1)), ..., (x_n, f(x_n))$, we are able to immediately get $f[x_0], ..., f[x_n]$. In particular we obtain c_0. Then we are able to use the formula above and obtain $f[x_0, x_1], f[x_1, x_2], ..., f[x_{n-1}, x_n]$. In particular we obtain c_1. Then, using the formula again, we can obtain $f[x_0, x_1, x_2], ..., f[x_{n-2}, x_{n-1}, x_n]$ via

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0},$$

for example. Inductively, we are able to compute all divided differences. It follows that we can obtain all the coefficients c_i's for Newton form.

Proof. Let $(x_0, f(x_0)), ..., (x_n, f(x_n))$ be given and suppose p_n interpolates all $n+1$ points with $\deg(p_n) \leqslant n$. Also suppose p_{n-1} interpolates all notes except x_n (so it interpolates n points) with $\deg(p_{n-1}) \leqslant n-1$. Further suppose $\tilde{p}_{n-1}$ interpolates all notes except x_0 (another polynomial with deg $\leqslant n-1$ that interpolates the last n points). What would the relation between p_n, p_{n-1}, and $\tilde{p}_{n-1}$ be? On one hand,

$$p_n(x) = p_{n-1}(x) + f[x_0, ..., x_n](x - x_0)(x - x_1)...(x - x_{n-1}),$$

and on the other hand,

$$p_n(x) = \tilde{p}_{n-1}(x) + f[x_1, ..., x_n, x_0](x - x_1)(x - x_2)...(x - x_n).$$

By the proposition above, $f[x_0, ..., x_n] = f[x_1, ..., x_n, x_0]$. Notice that

$$p_n(x) = \tilde{p}_{n-1}(x) + \frac{x - x_n}{x_n - x_0}[\tilde{p}_{n-1}(x) - p_{n-1}(x)].$$

It is clear that both LHS and RHS evaluate to 0 at x_i for $1 \leqslant i \leqslant n-1$. It is also clear that $p_n(x_n) = 0$. If $x = x_0$ then $p_n(x_0) = \tilde{p}_{n-1}(x) + (x_0 - x_n)/(x_n - x_0)[\tilde{p}_{n-1}(x_0) - p_{n-1}(x_0)] = p_{n-1}(x_0) = 0$ as well. Now what is the coefficient of x^n on both sides? That for the LHS is $f[x_0, ..., x_n]$. The one for the RHS is

$$\frac{x}{x_n - x_0} \cdot (f[x_1, x_2, ..., x_n] - f[x_0, x_1, ..., x_n]).$$

This proves the claim. □

Proposition 6.2.4

Let p be the polynomial of degree $\leqslant n$ that interpolates a function f on notes $x_0, ..., x_n$. If t is a point different from the nodes then

$$f(t) - p(t) = f[x_0, x_1, ..., x_n, t] \prod_{i=0}^{n} (t - x_i).$$

Proof in a nutshell: add a new datum $(t, f(t))$ and interpolate the original function at $n + 2$ nodes.

Theorem 6.2.5

If $f \in C^n[a, b]$ and if $x_0, ..., x_n$ are distinct points on $[a, b]$, then there exists a $\xi \in (a, b)$ satisfying

$$f[x_0, ..., x_n] = \frac{1}{n!} f^{(n)}(\xi).$$

In particular, the case $n = 1$ gives the MVT.

End of Course