# Numerical Analysis

Joseph Marziale

December 14, 2023

# Contents

# 1 August 29 to September 28

## 1.1 day01

Here we motivate numerical analysis in general. Equations such as $8x = e^{-x}$ have no closed form solution (try it!) despite a solution existing ($x = 0.1117\ldots$); therefore we need a way to approximate it. To do this we find the root $f = 0$ of the function

$$f = 8x - e^{-x} = 0 \leftrightarrow 8x = e^{-x}. \tag{1}$$

Newton's method says:

- pick an initial point $x_0$
- compute $f(x_0)$
- draw a linearization of the function at that point towards the $x$ axis (which requires its slope, $f'(x_0)$)
- define where the linearization touches the axis as $x_1$
- repeat: find $f(x_1)\ldots$

This is represented by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{2}$$

Our linearizations are guessing where the function *actually* touches the $x$ axis, which is the genuine root. Hopefully as we iterate we are getting closer and closer. Not true for many situations though. Valid questions in such cases are, when is convergence guaranteed? does the convergence yield a root? and, how fast is the convergence? and this is what numerical analysis is all about.

Computers are very good at doing repetitive things, so we use computers to do these iterations for us. Computers though possess finite precision of numbers. 64 sequential binary bits (basic objects that can contain either a 0 or 1) represent a machine number; therefore there can only be $2^{64}$ possible numbers. Therefore there must exist a largest machine number and a smallest one. What is the widest range of numbers we could possibly want to use for engineering? Let's say the range is ± Avogadro's number ($< 10^{24}$, $> -10^{24}$). Now since we only have $2^{64}$ numbers we must ask, how will they be spaced? Naively we could say, uniformly. But this means that the spacing of the numbers would be

$$\frac{10^{24} - -10^{24}}{2^{64}} = 108420.217249. \tag{3}$$

Therefore a sequence of machine numbers in our machine might be $0, \approx 108420, \approx 216840, \approx 325260, \ldots,$ meaning we wouldn't even be able to represent $1, 2, 3, \ldots$! So uniform spacing is bad. A better way (and what the industry uses) is to represent the 64 bits as

$$\pm 1.\{m\} \times 2^{c-1023}, \tag{4}$$

where

- first 1 bit goes to the sign ($0 \to +$, $1 \to -$)
- next 11 bits go to the exponent

$$(0 < c < \underbrace{2047}_{2^{11}-1})$$

$$\implies \text{ modified exponent: } (-1023 < \underbrace{c - 1023}_{modified\ exp} < 1024).$$

- last 52 bits go to the mantissa ($0 < m < 2^{52} - 1$)

This is a good relative spacing such that smaller numbers are closer together and larger numbers are further apart.

An example of a number is 123.91. Divide this by $2^6$ to get 1.93609375 (we modify the exponent on 2 until we get some 1.[...]). Therefore $123.91 = 1.\underbrace{93609375}_{m} \times 2^{\overbrace{6}^{c-1023}}$.

- Number is positive → first bit is 0.
- $c - 1023 = 6 \implies c = 1029 \leftrightarrow \underbrace{10000000101}_{11\ digits}$.
- $m = 0.93609375 \leftrightarrow 1110111110100011110101110000101000111101011100001010$

Therefore total machine number is

$$\underbrace{0}_{sign}\underbrace{10000000101}_{c}\underbrace{1110111110100011110101110000101000111101011100001010}_{m}$$

The hex conversion system is:

$0000 \leftrightarrow 0,$

$0001 \leftrightarrow 1, \quad \ldots,$

$1000 \leftrightarrow 8,$

$1001 \leftrightarrow 9,$

$1010 \leftrightarrow a,$

$1011 \leftrightarrow b,$

$1100 \leftrightarrow c,$

$1101 \leftrightarrow d,$

$1110 \leftrightarrow e,$

$1111 \leftrightarrow f.$

So the above machine number is converted to

$$|0100|0000|0101|1110|1111|1010|0011|1101|0111|0000|1010|0011|1101|0111|0000|1010$$
$$\leftrightarrow 405efa3d70a3d70a.$$

## 1.2 day02

In the earlier analysis we converted (**A**) a decimal (123.91) to (**B**) scientific notation ($1.93609375 \times 2^6$) to (**C**) iee754 binary (01000000010111101111101000111101011100001010001111010111 to (**D**) iee754 hex (405efa3d70a3d70a). But in (**B**) → (**C**) we implicitly converted $c = 1029, m = 0.93609375$ to binary individually without showing how. Now we show how for integers and fractions.

Given an integer such as $c = 1029$, divide by 2 and note the remainders:

$$1029 \div 2 = 514 \ R \ 1$$

$$514 \div 2 = 257 \ R \ 0$$

$$257 \div 2 = 128 \ R \ 1$$

$$128 \div 2 = 64 \ R \ 0$$

$$64 \div 2 = 32 \ R \ 0$$

$$32 \div 2 = 16 \ R \ 0$$

$$16 \div 2 = 8 \ R \ 0$$

$$8 \div 2 = 4 \ R \ 0$$

$$4 \div 2 = 2 \ R \ 0$$

$$2 \div 2 = 1 \ R \ 0$$

$$1 \div 2 = 0 \ R \ 1$$

$$0 \div 2 = 0 \ R \ 0$$

reading from the bottom up (omitting the trivial line $0 \div 2 = 0 \ R \ 0$ which would repeat infinitely), we get 10000000101.

Given a decimal such as 0.9 (follows same rules as $m = 0.93609375$ but easier to demonstrate), we multiply the fractional part by 2 and note the integer part:

$$0.9 \times 2 = (1).8$$

$$0.8 \times 2 = (1).6 \qquad \leftarrow$$

$$0.6 \times 2 = (1).2$$

$$0.2 \times 2 = (0).4$$

$$0.4 \times 2 = (0).8$$

$$0.8 \times 2 = (1).6 \qquad \leftarrow_{repeats\ starting\ from\ previous\ arrow}$$

reading from the top down we get $0.1\overline{11001} = 0.1(11001)(11001)(11001)(11001)\ldots$ .

Repeating decimals like this want to continue forever but the computer stops it at 52 digits which is the mantissa length. Machine epsilon $\epsilon_{mach} = 2^{-52}$ represents the relative spacing between numbers (i.e. the absolute spacing between 1 and the next highest binary number, $1.\underbrace{00000\ldots0001}_{52\ bin\ digits}$). If numbers are spaced as such, then the max relative error in representing a real number by a machine number is

$$\frac{\epsilon_{mach}}{2} := \delta. \tag{5}$$

Notice in all of this that there is no zero; the smallest positive number should be $c = 0, m = .00000 \to 1.00000 \times 2^{0-1023} = 2^{-1023}$. To solve this we just *manually let* $c = 0, m = 0$ imply $\pm 0$, not $2^{-1023}$. The effect of this is that it leaves a nonuniform spacing/ a hole at zero which can be remedied by denormalizing. For example consider a system with 2 mantissa length and 2 exponent length with 2 bias: the numbers $\leq 1$ before denormalization are:

$$0|00|00 = \cancel{1.00 \times 2^{-2}} =^{let} 0 \text{ (special rule)}$$

$$0|00|01 = 1.25 \times 2^{-2} = 0.3125$$

$$0|00|10 = 1.50 \times 2^{-2} = 0.375$$

$$0|00|11 = 1.75 \times 2^{-2} = 0.4375$$

$$0|01|00 = 1.00 \times 2^{-1} = 0.5$$

$$0|01|01 = 1.25 \times 2^{-1} = 0.625$$

$$0|01|10 = 1.50 \times 2^{-1} = 0.75$$

$$0|01|11 = 1.75 \times 2^{-1} = 0.875$$

$$0|10|00 = 1.00 \times 2^{0} = 1$$

Then to fix the hole at zero (0→0.3125) we do denormalization, which is just observing the spacing at the second-smallest order $\mathcal{O}(2^{-1})$ $(0.5 \leq x \leq 1)$ and applying it to the smallest order $\mathcal{O}(2^{-2})$ $(0 \leq x \leq 0.5)$:

$$00000 = \cancel{1.00 \times 2^{-2}} =^{let} 0 \text{ (special rule)}$$

$$00001 = \cancel{1.25 \times 2^{-2}} =^{let} 0.125 \text{ (denorm.)}$$

$$00010 = \cancel{1.50 \times 2^{-2}} =^{let} 0.25 \text{ (denorm.)}$$

$$00011 = \cancel{1.75 \times 2^{-2}} =^{let} 0.375 \text{ (denorm.)}$$

$$00100 = 1.00 \times 2^{-1} = 0.5$$

$$00101 = 1.25 \times 2^{-1} = 0.625$$

$00110 = 1.50 \times 2^{-1} = 0.75$

$00111 = 1.75 \times 2^{-1} = 0.875$

$01000 = 1.00 \times 2^{0} = 1.$

Now the hole at zero is not so large (0→0.125).

Similarly to manually setting zero: the largest possible number should be $c = \underbrace{1111\ldots11}_{bin}, m = \underbrace{111111\ldots111}_{bin}$, but we say that all numbers with this $c$ are just $\pm Inf$. Given that, the actual largest number possesses the bin exponent of $c = 11111111110$ corresponding to $\mathcal{O}(2^{2046-1023}) = \mathcal{O}(10^{307})$. It's big enough so we don't need the higher stuff anyway!

Remember, $\delta$ represents max relative round off error due to machine number approximation. That is

$$|\frac{fl(x) - x}{x}| \leq \delta = \frac{\epsilon_{mach}}{2} \tag{6}$$

where $fl(x)$ represents the machine approx'ed $x$.

The consquences of this are relevant to all machine computations because even if (say) $X, Y$ are machine numbers, this certainly does not mean $X + Y$ is a machine number, and so this sum will be approximated and will possess max relative round off error of

$$|\frac{X \bigoplus Y - X + Y}{X + Y}| \leq \delta, \qquad X + Y \neq 0 \tag{7}$$

where $\bigoplus$ indicates the machine sum and + indicates the analytical sum.

We've just supposed $X, Y$ are machine numbers but suppose we have generic $x, y \in \mathbb{R}$ and we're trying to compute $x \times y$. Then the several errors necessary to take into account are:

approximating $x$;

approximating $y$;

approximating their product.

Relative error becomes

$$\left|\frac{x(1+\delta_1)y(1+\delta_2)(1+\delta_3) - xy}{xy}\right| = \ldots = 3\delta \text{ (assuming } \delta_1 \approx \delta_2 \approx \delta_3 \approx \delta) \tag{8}$$

where $x(1 + \delta_1)$ is the machine approx'ed $x$, $y(1 + \delta_2)$ is the approx'ed $y$, and the whole first term is the approx'ed $xy$.

## 1.3 day03

Suppose we have a six digit system such that

$x = .500002 3\overline{0} \rightarrow fl(x) = .500002$

$y = .500001 1\overline{0} \rightarrow fl(y) = .500001$

$$\implies |\frac{fl(x) \ominus fl(y) - (x-y)}{x-y}| = |\frac{.00001 - .0000012}{.0000012}| \approx 17\%. \tag{9}$$

This is unacceptably high error, so as a remedy for situations such as this we turn to evaluating an approximation to the expression that is not subject to the catastrophic round off error, i.e. we do a Taylor approx. Taylor's theorem is

$$\begin{aligned} f(x) \approx f(x_0) + f'(x_0)(x-x_0) + f''(x_0)\frac{(x-x_0)^2}{2!} + f'''(x_0)\frac{(x-x_0)^3}{3!} + \dots \\ + f^{(n)}(x_0)\frac{(x-x_0)^n}{n!} + \underbrace{f^{(n+1)}(\xi)\frac{(x-x_0)^{n+1}}{(n+1)!}}_{R}, \end{aligned} \tag{10}$$

where $\xi \in [x_0, x]$. The last term is the truncation error $R$ and the other terms comprise the degree $n$ Taylor polynomial approximation. It can essentially be shown that for Taylor polys, the max round off error is the truncation error. For example the expression

$$(1 - e^p)c$$

is problematic for very small $p$, since the whole expression would be close to zero and thus small changes in $p$ lead to large changes in the round off error given a small number of rounding digits. But we can approximate $e^p$ as

$$e^p = 1 + p + \frac{p^2}{2!} + \underbrace{\frac{p^3 e^{\xi}}{3!}}_{R} \tag{11}$$

and for small $p$ we actually get decreasing $R$/less error. Therefore for small $p$ we prefer the Taylor approx but for large $p$ we prefer the closed form $e^p$.

## 1.4 day04

Various root finders (Newton, bisection, secant, etc.) trade off robustnness and speed. The most robust method is bisection:

Suppose $f \in C[a, b]$

$f(a)f(b) < 0$ s.t. one of $f(a), f(b)$ positive, the other negative, s.t. there is a root $z$ in the interval by IVT

Check sign of $f(c), c = (a + b)/2$ and replace either $a$ or $b$ by $c$ accordingly

- Repeat until desired precision reached

Then the $k$th calculation of $c$, i.e. $c_k$ is the $k$th approximation to the root $z$. This is because the size of the interval across iterations is

$$|a_k - b_k| = \frac{1}{2^k}|a_0 - b_0| \tag{12}$$

and with $c_k$ being in the middle of $a_k, b_k$, and with $z$ also being somewhere in the interval,

$$\implies |c_k - z| \leq \frac{1}{2}|a_k - b_k| = \frac{1}{2^{k+1}}|a_0 - b_0| \tag{13}$$

i.e. the max error is half the length of the $k$th interval.

If $z$ is the root of $f(x) = g(x) - x$ such that $f(z) = 0 = g(z) - z$, then we can stagger iteration of $g(z)$ and $z$ to our advantage in what is called fixed point root finding. This is best shown through example. If $f(x) = 2\cos x - 3x$ then

Root will be $f(z) = 0 = 2\cos z - 3z \implies z = \underbrace{\frac{2}{3}\cos z}_{g(z)}$

Do the iteration $g(z_k) = z_{k+1}$ until desired convergence:

say $z_0 = 0 \implies g(z_0) = \frac{2}{3}\cos 0 = z_1$
$z_1 = \frac{2}{3} \implies g(z_1) = \frac{2}{3}\cos\frac{2}{3} = z_2$
etc.

You can draw $g$ on the $x_k x_{k+1}$ plane (analogous to the $xy$ plane) and draw the iterative process like this:

pick $x_0$, start at $(x_0, 0)$

draw vert. line to $g$ to find $g(x_0) = x_1$

now you know $x_1$ so start at $(x_1, 0)$

draw vert. line to $g$ to find $g(x_1) = x_2$

now you know $x_2, \ldots$, etc.

## 1.5 day05

If $x_{k+1} = g(x_k)$ converges to $z$ such that $g(z) = z \to f = g(z) - z = 0$, then the order of the convergence is $q$ if

$$\underbrace{|x_{k+1} - z|}_{e_{k+1}} \leq c\,\underbrace{|x_k - z|}_{e_k},^q \qquad q = 1 \Leftrightarrow c < 1 \tag{14}$$

That is if $e_{k+1}$ (the interval/error between the $k+1$th guess and $z$) is the square root of $e_k$ (" " $k$th guess and $z$) then $q = 2$ and the sequence is second order convergent.

Note, $q$ is the first positive integer for which $g^{(q)}(z) \neq 0$ or if $q = 1$ then $|g'(z)| < 1$. Some examples:

- $g(x) = \frac{1}{5}x^2 + \frac{6}{5}$

  $g(z) = z \rightarrow z = \frac{1}{5}z^2 + \frac{6}{5} \rightarrow z = 2$

  $g'(x) = \frac{2}{5}x \rightarrow g'(z) = \frac{4}{5} \neq 0$

  $\implies q = 1$

- $g(x) = \frac{x}{2} + \frac{2}{x}$

  $g(z) = z \rightarrow z = \frac{z}{2} + \frac{2}{z} \rightarrow z = 2$

  $g'(x) = \frac{1}{2} - 2x^{-2} \rightarrow g'(z) = \frac{1}{2} - \frac{1}{2} = 0$

  $g''(x) = 4x^{-3} \rightarrow g''(z) = \frac{1}{2} \neq 0$

  $\implies q = 2$

Quadratic convergence doubles number of correct decimal digits $N_k$ per iteration $k$, while linear convergence increases $N_k$ by a potentially small constant $-\log c$:

- Definition: $N_k := -\log|e_k| \underbrace{\rightarrow}_{for\ example} e_k = .01 \implies N_k = -(-2) = 2$ correct digits in $k$th guess

- Linear:

$$|e_{k+1}| < c|e_k| \underbrace{\implies}_{take\ -log} -\log|e_{k+1}| \geq -\log|e_k| - \log c \implies N_{k+1} \geq N_k - \log c$$

- Quadratic:

$$|e_{k+1} < c|e_k|^2 \underbrace{\implies}_{take\ -log} -\log|e_{k+1}| \geq -2\log|e_k| - \log c \implies N_{k+1} \geq 2N_k - \log c$$

Cubic convergence would triple the number of correct digits, etc.

Newton's method artifically enforces nonlinear convergence by engineering a zero first derivative of $g$:

$$g(x_k) := x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{15}$$

At $z$, $f(z) = 0$ by def. of the root; substitute this in after computing $g'$ eval'ed at $z$:

$$g'(z) = 1 - \frac{f'(z)f'(z) - f(z)f''(z)}{f'^2(z)} = \cancel{1-1} + \frac{\cancelto{0}{f(z)}f''(z)}{f'^2(z)} \implies g'(z) = 0 \text{ unless } f'(z) = 0. \tag{16}$$

The function $g$ satisfies a Lipschitz condition in the region $G$ if $\exists\ L \geq 0$ such that

$$\frac{|g(x) - g(y)|}{|x_k - y_k|} \leq L \quad \forall x, y \in G. \tag{17}$$

$L$ is the slope of $g$ on the interval $[y, x]$ (rise/run). If $L > 1$ (slope of $g \in G \leq 1 \forall y, x$) then $g$ is said to be a contraction on $G$.

If $g$ is a contraction on $G$, then $\exists z \in G$ s.t. $z = g(z)$ and the sequence $x_{k+1} = g(x)$ converges to $z$ with error estimates

$$|x_k - z| \leq \frac{L^k}{1-L}|x_1 - x_0|,$$

$$|x_k - z| \leq \frac{L}{1-L}|x_k - x_{k-1}| \quad \text{(contraction mapping theorem/CMT).}$$

If we pick an interval $I = [z - \epsilon, z + \epsilon]$ and if we know $g'(z) < 1$ then $I$ is a small interval about $z$ and $g \in I$ satisfies CMT as it is a contraction.

## 1.6 day06

Newton requires the derivative $f'$, but if for some reason $f'$ is difficult to obtain, then we can use the secant method:

$$x_{k+1} = x_k - \frac{f(x_k)}{\left(\frac{f(x_{k-1}) - f(x_k)}{x_{k-1} - x_k}\right)} := x_k - \frac{f(x_k)}{\text{secant slope}} \tag{18}$$

$$\implies x_{k+1} = x_k - \frac{f(x_k)(x_{k-1} - x_k)}{f(x_{k-1}) - f(x_k)}. \tag{19}$$

Say we have initial guesses $x_0, x_1$. Newton's method says to use $f'(x_1)$ to get $x_2$. Instead, the secant method says to take the slope of the line that goes through $x_0, x_1$ as an approximation of the derivative at $x_1$. It converges superlinearly: $q = \gamma = \frac{1+\sqrt{5}}{2}$.

Superlinearly converging sequences $\{x_k\} \to z$ remove almost all error on each iteration:

$$|e_{k+1}| = c|e_k|^p, \quad p > 1 \underbrace{\implies}_{divide\ by\ e_k} \frac{|e_{k+1}|}{|e_k|} = c|e_k|^q, \quad q > 0 \tag{20}$$

$RHS \to 0$ due to convergence and $RHS = LHS$, so $LHS \to 0$, meaning $|e_k| \gg |e_{k+1}|$, i.e. most of the error goes away with further iteration. Therefore $|x_{k+1} - x_k| \approx |e_k|$. i.e. the amount of error solved for in the $k + 1$th step is almost the entire amount of error that existed to begin with. Thus iterate until

$$|x_{k+1} - x_k| \leq \text{ tolerance.} \tag{21}$$

# 2 September 28 to November 7

## 2.1 day07

Gaussian elimination and back substitution (GE, BS)

$$\begin{cases} x_1 + 2x_2 + x_3 = 3 \\ 2x_1 + 3x_2 - x_3 = -6 \\ 3x_1 - 2x_2 - 4x_3 = -2 \end{cases} \to \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & -1 \\ 3 & -2 & -4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3 \\ -6 \\ -2 \end{Bmatrix} \tag{22}$$

Augment

$$\begin{bmatrix} 1 & 2 & 1 & 3 \\ 2 & 3 & -1 & -6 \\ 3 & -2 & -4 & -2 \end{bmatrix} \tag{23}$$

Transform system into upper triangular using the operations

- swap two rows
- multiply row by nonzero scalar
- subtract a multiple of a row from another

then do back substitution to solve for $x_3, x_2$, then $x_1$. Cost of GE/BS is $\mathcal{O}(n^3)$ for $n \times n$ system matrix: three nested loops: (1) outer loop for each operation, (2) middle loop for each row, (3) inner loop for each column/element in that row.

## 2.2 day08

Matrix vector mult is $2n^2$ ops: $n$ mults and $n$ additions, $n$ times.

Errors in linear solve

- avoidable: swap rows such that diagonal element of the row being worked on is the largest in its column (GEPP)
- unavoidable: floating point arithmetic rounds off decimals in near-whole numbers ($3 - 7\epsilon \to 3$) but not in very small numbers ($\epsilon \to \epsilon$) leading to large error

$$\begin{bmatrix} \epsilon & 1 & 3 \\ 1 & 1 & 7 \end{bmatrix} \to \underbrace{\begin{bmatrix} \epsilon & 1 & 3 \\ 1 - \epsilon/\epsilon = 0 & 1 - 1/\epsilon & 7 - 3/\epsilon \end{bmatrix}}_{R_2 \leftarrow R_2 - R_1/\epsilon} \to \underbrace{\begin{bmatrix} \epsilon & 1 & 3 \\ 0 & 1 - \epsilon & 3 - 7\epsilon \end{bmatrix}}_{R_2 \leftarrow -R_2\epsilon} \tag{24}$$

$$\text{exact: } x_2 = (3-7\epsilon)/(1-\epsilon) \implies x_1 = [3-(3-7\epsilon)/(1-\epsilon)]/\epsilon = -4/(\epsilon-1) \approx 4 \tag{25}$$

$$\text{fp: } \approx \begin{bmatrix} \epsilon & 1 & 3 \\ 0 & 1 & 3 \end{bmatrix} \to x_2 = 3 \implies x_1 = (3-3)/\epsilon = 0 \tag{26}$$

## 2.3 day09

GEPP prevents inoperable rows (ones for which the diagonal element is zero) from attempting to propagate multiples of itself down the matrix (it cannot bring other terms to zero, for it is a zero). At stage $k$, swap row $k$ with row $l$ such that $|a_{lk}| = \max_j |a_{jk}|$.

When rescaling rows so that they are of the same order of magnitude, do so by powers of 2 so that only the exponent is changing and not the mantissa.

Vector norm

$$||x||_p = (\sum_i |x_i|^p)^{1/p} \tag{27}$$

Matrix inf norm i.e. max row sum

$$||A||_\infty = \max_i \sum_j |a_{ij}| \tag{28}$$

Matrix 1 norm i.e. max col sum

$$||A||_\infty = \max_j \sum_i |a_{ij}| \tag{29}$$

You can use whatever norm is convenient to you based on the application, since they are all equivalent in $\mathbb{R}^n, \mathbb{C}^n$.

If small changes in $A, b$ cause large changes in $x$ then the $Ax = b$ is called ill-conditioned.

$$(A + \delta A)(x + \delta x) = b + \delta b \tag{30}$$

If

$$||\delta A||||A^{-1}|| < 1 \text{ ($\delta A$ not too large)} \tag{31}$$

Then

$$\frac{||\delta x||}{||x||} \leq \frac{\kappa(A)}{1 - ||\delta A||||A^{-1}||}(\frac{||\delta b||}{||b||} + \frac{||\delta A||}{||A||}) \tag{32}$$

in which the condition number of $A$ is

$$\kappa(A) = ||A||||A{-}1|| \tag{33}$$

and a large $\kappa(A)$ implies an ill conditioned system matrix.

each column of $A^{-1} = X = [X_1 \; X_2 \; \dots \; X_n]$ requires the solution to a linear system

$$AX_1 = I_1 = \begin{Bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}, \quad AX_2 = I_2 = \begin{Bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{Bmatrix}, \quad AX_n = I_n = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{Bmatrix} \tag{34}$$

since this is expensive you can estimate $\kappa(A)$ without forming $A^{-1}$.

Wikkinson round off error in GE

$$\frac{|x_{gepp} - x}{|x|} \leq 4n^2\kappa(A)\rho\epsilon_{mach} \tag{35}$$

in which growth factor

$$\rho = \max_{k,i,j} |a_{ij}^{(k)}| / \max_{i,j} |a_{ij}| \leq 2^{n-1} \tag{36}$$

is the max element across all stages of the matrix during GEPP, relative to the max element in the initial stage.

Hilbert matrix is very ill conditioned

$$H_{ij} = 1/(i+j-1) \leftrightarrow \begin{bmatrix} 1 & 1/2 & \dots & 1/n \\ 1/2 & 1/4 & \dots & 1/(n+1) \\ \vdots & & & \\ 1/n & 1/(n+1) & \dots & 1/(2n-1) \end{bmatrix} \tag{37}$$

GE as LU decomposition

$$Ax = b \to LUx = b \to \begin{cases} Lc = b & \mathcal{O}(n^2) \\ Ux = c & \mathcal{O}(n^2) \end{cases} \tag{38}$$

Total work is $\mathcal{O}(n^3)$.

Get $U$ out of $A$ through GE; each step of GE represented as a matrix transformation

$$M_6 M_5 \dots M_2 M_1 A = U \tag{39}$$

$$M_1, M_2, M_3, M_4, M_5, M_6 = \begin{bmatrix} 1 & & & \\ -m_1 & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1 & & \\ -m_2 & & 1 & \\ & & & 1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ -m_3 & & & 1 \end{bmatrix}, \tag{40}$$

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & -m_4 & 1 & \\ & & & 1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & -m_5 & & 1 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -m_6 & 1 \end{bmatrix} \tag{41}$$

then do

$$A = LU = L(M_6 M_5 \dots M_2 M_1 A) \to L = M_1^{-1} M_2^{-2} \dots M_5^{-1} M_6^{-1} \tag{42}$$

where the inverse matrices represent the inverse set of operations done on $A$ to get $U$. I.e.

$$M_4 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -m_4 & 1 & \\ & & & 1 \end{bmatrix} \to M_4^{-1} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & m_4 & 1 & \\ & & & 1 \end{bmatrix} \tag{43}$$

## 2.4 day11

Overdetermined linear systems $A_{m \times n} x_{n \times 1} = b_{m \times 1}, \quad m \gg n$ (many more rows than columns) typically have no true solution. Normal equations: we minimize residual

$$r_{m \times 1} = Ax - b =^{set} 0 \tag{44}$$

$$0 = A^T_{n\times m} r_{m\times 1} = A^T(Ax-b) \to A^TAx = A^Tb \tag{45}$$

$A^TA$ is ill conditioned → unstable.

QR (orthogonal; upper/right triangular) factorization: factorize $A$ as

$$A_{m\times n} = Q_{m\times m} \begin{bmatrix} R_{n\times n} \\ 0_{m-n\times n} \end{bmatrix}_{m\times n} \tag{46}$$

Useful because orthogonal $Q$ preserves 2-norm:

$$||Qy||_2 = \sqrt{Qy\cdot Qy} = \sqrt{y^TQ^TQy} = \sqrt{y^Ty} = y\cdot y = ||y||_2 \tag{47}$$

Residual

$$0 =^{set} ||r||_2^2 = ||Ax-b||_2^2 = ||Q^T(Ax-b)||_2^2 = ||Q^T(Q\begin{bmatrix} R \\ 0 \end{bmatrix} x - b)||_2^2 \tag{48}$$

$$= ||Q^TQ\begin{bmatrix} R \\ 0 \end{bmatrix} x - Q^Tb||_2^2 = || \begin{bmatrix} R \\ 0 \end{bmatrix} x - Q^Tb||_2^2 \tag{49}$$

$$= || \begin{bmatrix} Rx \\ 0 \end{bmatrix} - \begin{bmatrix} c \\ d \end{bmatrix} ||_2^2 = ||Rx-c||_2^2 + \overset{\text{don't care}}{\cancel{||d||_2^2}} \tag{50}$$

$$\implies 0 =^{set} Rx - c \to Rx = c \tag{51}$$

in which $c$ is the vector containing the first $n$ components of $Q^Tb$.

Get $R0$ out of $A$; each step represented as an orthogonal matrix transformation

$$H_n \dots H_2H_1A = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{52}$$

Then do

$$A = Q\begin{bmatrix} R \\ 0 \end{bmatrix} = Q(H_n \dots H_2H_1A) \to Q = H_1^TH_2^T \dots H_n^T \tag{53}$$

we have $n$ steps because the first $n$ columns of $A$ are manipulated to obtain $R_{n\times n}$:

$$[A]_{m\times n} \xrightarrow{H_1} \begin{bmatrix} r_{11} & a_{12} & a_{13} & \dots \\ 0 & a_{22} & a_{23} & \dots \\ 0 & a_{32} & a_{33} & \dots \\ \vdots & & & \\ \underbrace{0}_{col1} & a_{n2} & a_{n3} & \dots \end{bmatrix} \xrightarrow{H_2} \begin{bmatrix} r_{11} & r_{12} & a_{13} & \dots \\ 0 & r_{22} & a_{23} & \dots \\ 0 & 0 & a_{33} & \dots \\ \vdots & \vdots & & \\ 0 & \underbrace{0}_{col2} & a_{n3} & \dots \end{bmatrix} \xrightarrow{\cdots} \xrightarrow{H_n} \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{54}$$

Find $H_i$:

- build $H_i$s using projections of the form

  $$P = vv^T/v^Tv. \tag{55}$$

  given $\vec{u}, \vec{v}$, projection $P\vec{u} = \text{proj}_{\vec{v}}\vec{u}$ is the shadow of $\vec{u}$ on $\vec{v}$.

- let the column we are manipulating be $z$. We want

  $$Hz = w \tag{56}$$

  in which $w = \{w_1, 0, 0, \dots, 0\}^T$.

- $H$ is 2-norm preserving $\rightarrow \pm||z||_2 = w_1$. So we know what we want $w_1 \leftrightarrow w$ to be.

- Let $v := w - z$. Then the triangle $vwz$ is isosceles ($||w||_2 = ||z||$). Then the projection/shadow of $z$ on $v$ is one half of $v$. Therefore $v$ represents the process of $H$ taking us from $z$ to $(z - Pz - Pz) = w$

  $$\implies w = (I - 2P)z \tag{57}$$

  $$H = I - 2P = I - 2(vv^T/v^Tv), \qquad v = w - z \tag{58}$$

To prove $H$ is orthogonal

$$H^TH = (I-2P)^T(I-2P) = I - 2P - 2P^T + P^TP \tag{59}$$

$P$ symmetric: $P^T = (vv^T/v^Tv)^T = v^{T^T}v^T/v^Tv = P$. Thus

$$H^TH = I - 4P + 4P^2 \tag{60}$$

$$= \delta_{ij} - 4v_iv_j/v_kv_k + 4v_i \cancel{v_lv_l}\, v_j/v_kv_k \cancel{v_rv_r} = \delta_{ij} = H^TH \rightarrow H^T = H^{-1}. \tag{61}$$

That takes care of $H_1$ to handle the first column. For column 2,

$$H_2 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & & & & \\ 0 & & & \hat{H}_2 & \\ \vdots & & & & \\ 0 & & & & \end{bmatrix} \tag{62}$$

in which $\hat{H}_2$ is computed in the same manner as $H_1$, except applied to the submatrix of $A$ without first row and column.

$$H_3 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & \hat{H}_3 & \\ 0 & 0 & & & \end{bmatrix} \tag{63}$$

etc. This is called Householder factorization.

## 2.5 day12

$QR$ useful for overdetermined systems because we minimize $Rx = c$, where $c$ is the first $n$ components of $Q^Tb$. An example of an overdetermined system is the linear-ish data set $(t_i, y_i)$ s.t. $y_i \approx x_1 t_i + x_2$, $i = 1, \dots m$

$$\implies \begin{bmatrix} t_1 & 1 \\ t_2 & 1 \\ \vdots & \vdots \\ t_m & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{Bmatrix} \tag{64}$$

with this we can factor $A$ as $QR$ and solve

$$R \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \left\{ \text{first 2 elements of } Q^T y \right\}. \tag{65}$$

Quadratic-ish data set $(t_i, y_i)$ s.t. $y_i \approx x_1 t_i^2 + x_2 t_i + x_3$

$$\implies \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ \vdots & \vdots & \vdots \\ t_m^2 & t_m & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{Bmatrix} \tag{66}$$

do the same thing.

Polynomial approximation

Weierstrass proof: Bernstein polynomials $\{b_m(h, x)\}_{m=0,1,2,\dots,\infty}$ in which

$$b_m(h, x) = \sum_{k=0}^{m} h\left(\frac{k}{m}\right) \binom{m}{k} x^k (1-x)^{m-k} \tag{67}$$

necessarily converge to analytical polynomial function such that $b_m(h, x) \rightarrow^{m=\infty} h(x)$. They prove that any function can be approximated, but they are impractical/expensive. We turn to other methods...

## 2.6 day13

Alternatives to Bernstein polynomials

- Taylor polynomials ($f \in C^{n+1}[a, b]$, $x_0 \in [a, b]$, $\xi \in [x_0, x]$)

$$p_n = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k \tag{68}$$

has error

$$R_n(x) = \frac{f^{(k+1)}(\xi)}{(k+1)!} (x - x_0)^{k+1} \tag{69}$$

- Degree $n$ interpolants.

In general, the function $p(x)$ interpolates $(x_j, y_j = f(x_j))$ if $p(x_j) = y_j \forall j$.

Lagrange basis

$$l_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j} \tag{70}$$

Properties

- $l_k$ is of degree $n$.
- $l_k(x_j) = \delta_{kj}$

Then the Lagrange polynomial

$$p(x) = \sum_{k=0}^{n} y_k l_k(x) \tag{71}$$

interpolates the data: $p(x_j) = 0 + 0 + \ldots + 0 + y_j + 0 + \ldots + 0$.

In general a polynomial interpolant takes the form

$$p(x) = \sum_{k=0}^{n} c_k b_k(x) \tag{72}$$

in which $b_k(x)$ are the basis functions and $c_k$ are the coefficients.

Lagrange basis: pros:

- coefficients require no computation:$c_k = y_k$

cons:

- inefficient to evaluate
- need to recompute from scratch if new node is added

Monomial basis:

$$b_k(x) = x^k \to \{b_k\} = \{1, x, x^2, \ldots, x^n\} \tag{73}$$

interpolation requires $p(x_j) = \sum_{k=0}^{n} b_k(x_j) c_k = y_j$

$$\begin{bmatrix} b_0(x_0) & b_1(x_0) & \ldots & b_n(x_0) \\ b_0(x_1) & b_1(x_1) & \ldots & b_n(x_1) \\ \vdots & & & \\ b_0(x_n) & b_1(x_n) & \ldots & b_n(x_n) \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \vdots \\ y_n \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{Bmatrix} \tag{74}$$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ \vdots \\ y_n \end{Bmatrix} = \begin{Bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{Bmatrix} \leftrightarrow Ac = y \tag{75}$$

pros:

- basis is simple

cons:

- ill conditioned $A$ → large $\kappa(A)$
- lots of computation towards $c_k$

Newton basis

$$\phi_0(x) = 1 \tag{76}$$

$$\phi_1(x) = (x - x_0)\phi_0(x) \tag{77}$$

$$\phi_2(x) = (x - x_1)\phi_1(x) \tag{78}$$

$$\phi_n(x) = (x - x_{n-1})\phi_{n-1}(x) \tag{79}$$

pros:

- adding a new data point does not lead to recomputing existing basis
- tabular scheme of getting $c_k$s available

## 2.7 day14

Newton basis cont.

$$p(x_j) = \sum_{k=0}^{n} c_k \phi_k(x_j) = y_j \tag{80}$$

$$\begin{bmatrix} 1 & \cancel{x_0 - x_0} & \cancel{(x_0 - x_0)}(x_n - x_1) & \dots \\ 1 & x_1 - x_0 & (x_1 - x_0)\cancel{(x_1 - x_1)} & \dots \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & \dots \\ \vdots & & & \\ 1 & x_n - x_0 & (x_n - x_0)(x_n - x_1) & \dots \end{bmatrix} \tag{81}$$

upper right triangle is zeros → lower triangular Newton basis.

If $\{y_i\}$ is all we are given then technically we are doing a perfect job in that $p(x_j) = y_j \forall j$. But if $\{y_j\}$ are actually $\{f(x_j)\}$ for some function $f$ then the question arises, how well are we doing between the nodes? I.e., find the magnitude of

$$||p - f||_\infty = \max_{x \in [a,b} |p(x) - f(x)| \tag{82}$$

Taylor polynomial: error bound is

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \tag{83}$$

Polynomial interpolant: error bound is

$$\text{error}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_n) \tag{84}$$

Without knowing our $f$, all we can control to minimize error is our choice of $x_i$. We want to minimize

$$W(x) = (x - x_0)(x - x_1) \dots (x - x_n) = \prod_{i=0}^{n} (x - x_i) \tag{85}$$

The optimal choice is the Chebeshev nodes

$$x_i = \cos(\frac{2i+1}{n+1} \frac{\pi}{2}), \quad i = 0, 1, \dots, n. \tag{86}$$

More nodes does not always lead to a better interpolant. 2 uniformly spaced points are better than 3 for the Runge function

$$r(x) = \frac{1}{1 + 25x^2}. \tag{87}$$

But,

- Uniform nodes: $||p - r||_\infty \to \infty$ as $n \to \infty$ (bad)
- Chebyshev nodes: $||p - r||_\infty \to 0$ as $n \to \infty$ (good)

## 2.8 day15

Piecewise polynomial interpolation

Linear piecewise interpolant: subinterval between points $(x_i, y_i), (x_{i+1}, y_{i+1})$

$$l_i(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i) \tag{88}$$

We can construct a piecewise basis

$$\phi(x) = \sum c_i \phi_i(x) \tag{89}$$

where $c$ are the coefficients and $\phi_i$ are "hat functions" that go to 1 at $y_i$ and go to zero at $y_{j \neq i}$.

Error of PW interpolant if $y_i = f(x_i)$:

- $f \in C^2[a, b] \to ||\phi - f||_\infty \leq h^2 ||f''||_\infty / 8$
- $f \in C^1[a, b] \to ||\phi - f||_\infty \leq h ||f'||_\infty / 2$

in which $h = \max_i |x_{i+1} - x_i|$ is the largest node spacing.

Piecewise cubic functions/cubic splines

Subintervals are cubic functions

$$\phi_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \tag{90}$$

Motivation for cubic splines: elastic beam satisfies $y^{(4)}(x) = w(x) =$ load force. No load $\implies y^{(4)} = 0 \to y$ is cubic. With a point load, $y^{(3)}$ will jump at each of the points where load is applied; $y^{(2)}, y^{(1)}$ are always continuous.

This motivates seeking a $\phi \in C^2[a, b]$ that is cubic on each subinterval.

Degrees of freedom vs. constraints: suppose we have $n + 1$ nodes $0, \dots, n$, thus $n$ subintervals $0, \dots, n - 1$

- dofs: 4 coefficients at each subinterval = $4n$ dofs.
- constraints: $4(n-2)$
  - $\phi_i(x_i) = y_i$, $i = 0, \dots, n-1$
  - $\phi_i(x_{i+1}) = y_{i+1}$, $i = 0, \dots, n-1$
  - $\phi'(x_{i+1}) = \phi_{i+1}(x_{i+1)}$, $i = 0, \dots, n-2$
  - $\phi''(x_{i+1}) = \phi_{i+1}(x_{i+1)}$, $i = 0, \dots, n-2$

We must choose 2 more constraints

- Natural BCs: $\phi''(x_0) = 0$, $\phi''(x_n) = 0$
- Clamped BCs: specify $y'$ at the ends

Linear system

$$\begin{bmatrix}
1 & \dots & & x_0 & \dots & & x_0^2 & \dots & & x_0^3\dots & \\
 & 1 & \dots & & x_1 & \dots & & x_1^2 & \dots & x_1^3\dots & \\
 & & 1 & \dots & & x_2 & \dots & & x_2^2 & \dots & x_2^3\dots \\
\vdots & & & & & & & & & & \\
1 & \dots & & x_1 & \dots & & x_1^2 & \dots & & x_1^3\dots & \\
 & 1 & \dots & & x_2 & \dots & & x_2^2 & \dots & x_2^3\dots & \\
 & & 1 & \dots & & x_3 & \dots & & x_3^2 & \dots & x_3^3\dots \\
\vdots & & & & & & & & & & \\
0 & 0 & \dots & 1 & -1 & \dots & 2x_1 & -2x_1 & \dots & 3x_1^2, -3x_1^2 & \\
\vdots & & & & & & & & & & \\
0 & 0 & \dots & 0 & 0 & \dots & 2 & -2 & \dots & 6x_1, -6x_1 & \\
\vdots & & & & & & & & & & \\
\text{extra} & & & & & & & & & &
\end{bmatrix}
\left\{\begin{matrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{n-1} \end{matrix}\right\}
=
\left\{\begin{matrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{matrix}\right\}
\tag{91}$$

- First $n$ columns of $y : \phi_i(x_i) = y_i$, $i = 0, \dots, n-1$

$$a_i + b_i x_i + c_i x_i^2 + d_i x_i^3 = y_i \tag{92}$$

- Second $n$ columns of $y : \phi_i(x_{i+1}) = y_{i+1}$, $i+1 = 1, \dots, n$

$$a_i + b_i x_{i+1} + c_i x_{i+1}^2 + d_i x_{i+1}^3 = y_{i+1} \tag{93}$$

- Next/third $n-1$ columns of $y$ : $\phi_i'(x_{i+1}) - \phi_{i+1}'(x_{i+1}) = 0$, $i = 0, \ldots, n-2$

$$b_i - b_{i+1} + 2c_i x_{i+1} - 2c_{i+1} x_{i+1} + 3d_i x_{i+1}^2 - 3d_{i+1} x_{i+1}^2 = 0 \tag{94}$$

- Next/fourth $n-1$ columns of $y$ : $\phi_i''(x_{i+1}) - \phi_{i+1}''(x_{i+1}) = 0$, $i = 0, \ldots, n-2$

$$2c_i - 2c_{i+1} + 6d_i x_{i+1} - 6d_{i+1} x_{i+1} = 0 \tag{95}$$

- Final two columns: extra BCs
  - natural

$$2c_0 + 6d_0 x_0 = 0 \tag{96}$$

$$2c_n + 6d_n x_n = 0 \tag{97}$$

  - clamped

$$b_0 + 2c_0 x_0 + 3d_0 x_0^2 = C_1 \tag{98}$$

$$b_n + 2c_n x_n + 3d_n x_n^2 = C_2 \tag{99}$$

Bezier/PW cubic parametric curves

$$B(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{Bmatrix} a + bt + ct^2 + dt^3 \\ e + ft + gt^2 + ht^3 \end{Bmatrix}, \ t \in [0,1] \tag{100}$$

$$B(t) = s^3 P_0 + 3s^2 P_1 + 3st^2 P_2 + t^3 P_4, \tag{101}$$

$$s = 1 - t \tag{102}$$

$$B(0) = P_0, \quad B(1) = P_3, \quad P'(0) = 3(P_1 - P_0), \quad P'(1) = 3(P_3 - P_2) \tag{103}$$

$P_i$ are points $(x_i, y_i)$. Start point is $P_0$, end point is $P_3$, and $P_1, P_2$ help determine slope. These help represent arbitrary curves in a plane instead of just graphs of functions.

## 2.9 day16

Let

$$\omega_j(x) = e^{ijx} = (e^{ix})^j, \quad i = \sqrt{-1}, \quad j = -k, \ldots, 0, \ldots, k. \tag{104}$$

Let

$$W_{2k} = \{\omega_{-k}, \ldots, \omega_0, \ldots, \omega_k\} \tag{105}$$

Any $\omega_j$ is orthogonal to all of $W_{2k}$ in that

$$\int_0^{2\pi} \omega_j(x)\overline{\omega_l(x)} dx := (\omega_j, \omega_l) = 2\pi\delta_{jl}, \quad \delta_{jl} = \text{ Kronecker delta}, \quad c = a+bi \implies \bar{c} = a-bi. \tag{106}$$

All of $\omega_j \in W_{2k}$ are contained in $V = C[0, 2\pi]$, $V$ being the set of all continuous functions on the interval $0, 2\pi$.

Say we have $f \in V$ we want to approximate with

$$g_k(x) = \sum_{-k}^{k} \alpha_j \omega_j(x) \qquad \in W_{2k}. \tag{107}$$

If we were to do so perfectly (such that $g_k = f$), then

$$(f, \omega_l) = (g_k, \omega_l)$$

$$= (\sum_{-k}^{k} \alpha_j \omega_j, \omega_l) = \underbrace{\alpha_l(\omega_l, \omega_l)}_{only\ the\ lth\ term\ doesn't\ vanish} = 2\pi\alpha_l$$

$$\implies \alpha_l = \frac{1}{2\pi}(f, \omega_l) := \frac{1}{2\pi}\int_0^{2\pi} f(x) e^{ilx} dx, \quad l = -k, \dots, 0, \dots, k.$$

This is how to calculate $\alpha_l$ for each $\omega_l \in W_{2k}$; in summary

$$f(x) \approx g_k(x) = \sum_{j=-k}^{k} \alpha_j e^{ijx}, \qquad \alpha_j = \frac{1}{2\pi}\int_0^{2\pi} f(x) e^{ijx} dx \tag{108}$$

Separate $j = 0 \to e^{i0x}$ term and make the summation more efficient

$$f(x) \approx g_k(x) = \alpha_0 + \sum_{j=1}^{k} (\alpha_j e^{ijx} + \alpha_{-j} e^{-ijx}) \tag{109}$$

Recall $e^{ijx} = \cos jx + i \sin jx$, and that $i \sin(-jx) = -i \sin jx$. Substituting

$$f(x) \approx g_k(x) = \alpha_0 + \sum_{j=1}^{k} \Big[\alpha_j(\cos jx + i \sin jx) + \alpha_{-j}(\cos jx - i \sin jx)\Big]$$

$$= \alpha_0 + \sum_{j=1}^{k} \Big[\underbrace{(\alpha_j + \alpha_{-j})}_{a_j} \cos jx + \underbrace{i(\alpha_j - \alpha_{-j})}_{b_j} \sin jx\Big]$$

$$\implies \boxed{f(x) \approx g_k(x) = \alpha_0 + \sum_{j=1}^{k} [a_j \cos jx + b_j \sin jx]}, \tag{110}$$

$$\boxed{a_j = \alpha_j + \alpha_{-j}, \quad b_j = \alpha_j - \alpha_{-j}, \quad \alpha_k = \int_0^{2\pi} f(x) e^{ikx} dx}, \quad \begin{cases} e^{ipx} = \cos px + i \sin px \\ e^{-ipx} = \cos px - i \sin px \\ \cos(-px) = \cos px \\ i \sin(-px) = -i \sin px. \end{cases} \tag{111}$$

How does accuracy improve as $k \to \infty$? If $f$ is PW continuous on $[0, 2\pi]$, and

$$f^{(p)}(0) = f^{(p)}(2\pi), \quad p = 0, 1, \dots, n-1 \tag{112}$$

then

$$||f - g_k||_\infty = \mathcal{O}(1/k^{n-1}). \tag{113}$$

The above is for continuous $f$, but suppose we have discrete data $x_k = 2\pi\frac{k}{n}$, $k = 0, 1, \ldots, n-1$, and $y_k$. Then $p$ interpolates the data if

$$\boxed{p(x) = \sum_{j=0}^{n-1} c_j e^{ijx}, \quad c_j = \frac{1}{n}\sum_{k=0}^{n-1} y_k \omega^{jk}, \quad \omega = e^{-i2\pi/n}, \quad \{x_q, y_q\}_{q=0,\ldots,n-1}}, \Big\{\overline{\omega} = \tfrac{1}{\omega} \tag{114}$$

Then (index notation)

$$c_j = W_{jk}y_k, \quad W_{jk} = \frac{1}{n}\omega^{jk}$$

$$\Leftrightarrow \left\{\begin{matrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{matrix}\right\} = \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2(n-1)} \\ \vdots & \vdots & & & \\ 1 & \omega^{n-1} & & & \omega^{(n-1)^2} \end{bmatrix} \tag{115}$$

$W_{jk}$ known as discrete Fourier transform/DFT matrix, having the property $W_{jq}W_{qk}^{*\leftarrow conj\ transpose} = n\delta_{jk}$.

# 3 November 7 to December 14

## 3.1 day17

Note, different sources use different $W$:

- $F_{ack} = \frac{1}{n} W \rightarrow F_{ack}^{-1} = \overline{W}$
- $F_u = \frac{1}{\sqrt{n}} W \rightarrow F_u^{-1} = \frac{1}{\sqrt{n}} \overline{W} \rightarrow F_u F_u^* = I$
- $F_{numpy} = W \rightarrow F_{numpy}^{-1} = \frac{1}{n} \overline{W}$.

Given discrete $x_k, y_k$, $k = 0, 1, \ldots, n-1$, the highest frequency oscillation supportable by the grid is one in which the nodes represent successive peaks/troughs, called the Nyquist frequency, which has wave number $\kappa = \frac{n}{2}$ since $n/2$ waves fit in $[x_0, x_n]$ (try it on say $n = 8$). And as it turns out our trig interpolations on the same grid will look similarly to the Nyquist freq, i.e. it will oscillate up and down, which makes no sense. ...

## 3.2 day18

For large number of discrete data $n = 162,000$ (say), DFT matrix multiplication $c_j = (F_n)_{jk} y_k$ not feasible, but we can use the more feasible FFT:

- Recalling $(F_n)_{jk} = \omega^{jk}$, the shorthand for the exponents of the elements is

$$F_8 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 2 & 4 & 6 & 8 & 10 & 12 & 14 \\ 0 & 3 & 6 & 9 & 12 & 15 & 18 & 21 \\ 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 \\ 0 & 5 & 10 & 15 & 20 & 25 & 30 & 35 \\ 0 & 6 & 12 & 18 & 24 & 30 & 36 & 42 \\ 0 & 7 & 14 & 21 & 28 & 35 & 42 & 49 \end{bmatrix} \tag{116}$$

- Divide odd index columns by $k = 1$ column (subtracting exponents)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 4 & 4 & 6 & 6 \\ 0 & 0 & 4 & 4 & 8 & 8 & 12 & 12 \\ 0 & 0 & 6 & 6 & 12 & 12 & 18 & 18 \\ 0 & 0 & 8 & 8 & 16 & 16 & 24 & 24 \\ 0 & 0 & 10 & 10 & 20 & 20 & 30 & 30 \\ 0 & 0 & 12 & 12 & 24 & 24 & 36 & 36 \\ 0 & 0 & 14 & 24 & 28 & 28 & 42 & 42 \end{bmatrix} \tag{117}$$

- Mod by 8, not changing the values since $\omega^8 = 1$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 4 & 4 & 6 & 6 \\ 0 & 0 & 4 & 4 & 0 & 0 & 4 & 4 \\ 0 & 0 & 6 & 6 & 4 & 4 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 4 & 4 & 6 & 6 \\ 0 & 0 & 4 & 4 & 0 & 0 & 4 & 4 \\ 0 & 0 & 6 & 6 & 4 & 4 & 2 & 2 \end{bmatrix} \tag{118}$$

Now upper half of even columns = upper half of odd columns = lower half of even columns = lower half of odd columns =

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 6 \\ 0 & 4 & 0 & 4 \\ 0 & 6 & 4 & 2 \end{bmatrix} \sim F_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 4 & 6 \\ 0 & 3 & 6 & 9 \end{bmatrix} \tag{119}$$

We conclude

$$F_8 y = \frac{F_4 y_{even:\ 0,2,4,6} + \begin{Bmatrix} \omega^0 \\ \omega^1 \\ \omega^2 \\ \omega^3 \end{Bmatrix}^{compensating\ for\ subtraction\ of\ first\ col} \qquad F_4 y_{odd:\ 1,3,5,7}}{F_4 y_{even} + \begin{Bmatrix} \omega^4 \\ \omega^5 \\ \omega^6 \\ \omega^7 \end{Bmatrix} F_4 y_{odd}} \tag{120}$$

where the line is not meant to indicate a fraction but a separation of submatrices. Then we can apply the same technique to $F_4$ in terms of $F_2$, and so on until in terms of $F_1 = [1]$. Operation cost of FFT is almost linear with $n$, much better than $\mathcal{O}(n^2)$ of DFT.

## 3.3 day19

Ways to approximate derivatives

- Symbolically:

  pros: exact, and pretty close in floating point arithmetic

  cons: very inefficient

- Finite differences:
  - pros: very general, don't need a formula
  - truncation error, round off error

Finite differences are derived from the Taylor approx of $f(x_0)$ about $h$. Forward finite difference:

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}f'(\xi) \implies f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} + \mathcal{O}(h). \quad (121)$$

With small $h$ we get round off error, but with large $h$ we get Taylor truncation error, so there is a limited region with small error.

Central finite difference:

$$\begin{cases} f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \mathcal{O}(h^3) \\ f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \mathcal{O}(h^3) \end{cases} \implies \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^3) = f'(x). \quad (122)$$

Automatic differentiation does not possess truncation or roundoff error. AD:

> Say we have 2 functions $f, g$ and $h = fg$ and we want $h'(a)$, $a$ being a number.
>
> Define two objects $< f(a), f'(a) >, < g(a), g'(a) > \iff < f_{val}, f_{der} >, < g_{val}, g_{der} >$
>
> Then $< h_{val}, h_{der} >=< f_{val}, f_{der} > \times < g_{val}, g_{der} >=< f_{val}g_{val}, f_{val}g_{der} + g_{val}f_{der} >$.

Other arithmetic operations

- Addition/subtraction: $< f_v, f_d > \pm < g_v, g_d >=< f_v \pm g_v, f_d \pm g_d >$
- Multiplication: $< f_v, f_d > \times < g_v, g_d >=< f_vg_v, f_vg_d + g_vf_d >$
- Division:
$$\frac{< f_v, f_d >}{< g_v, g_d >} =< \frac{f_v}{g_v}, \frac{g_vf_d - f_vg_d}{g_v^2} >$$
- Trig: $\sin(< f_v, f_d >) =< \sin f_v, f_d \cos f_v >$ (similar for other trig)

## 3.4 day21

Forward mode AD on $\frac{x^2}{x+7}$ at $x = 3$:

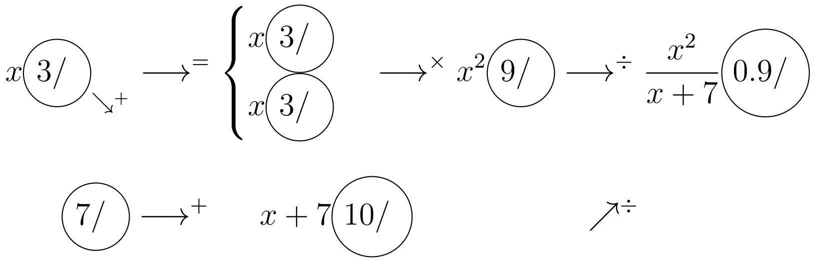
> Create object $< x_v, x_d >=< 3, 1 >$
>
> Do:
>
> $x^2 =< x_v, x_d > \times < x_v, x_d >$
>
> $x + 7 =< x_v, x_d > + < 7, 0 >$
>
> $< first,\ result > / < second,\ result >$.

We can extend this algebra of course to higher order derivatives. Or we can do multiple variables:

$$< f, \nabla f >< g, \nabla g >=< fg, f\nabla g + g\nabla f > .$$

Reverse mode AD (used in machine learning) on $\frac{x^2}{x+7}$ at $x = 3$:

- Create variable $x$ 3/, weight 7/
- Copy image twice for $x^2$ and create web of forward operations:



- Let the result $(\frac{x^2}{x+7})$ be $y$
- First backpropagation

  Let $(\frac{x^2}{x+7}) = x_k = x_i/x_j = 0.9$

  Let $(x^2) = x_i = 9$, let $(x+7) = x_j = 10$

  Fill in:

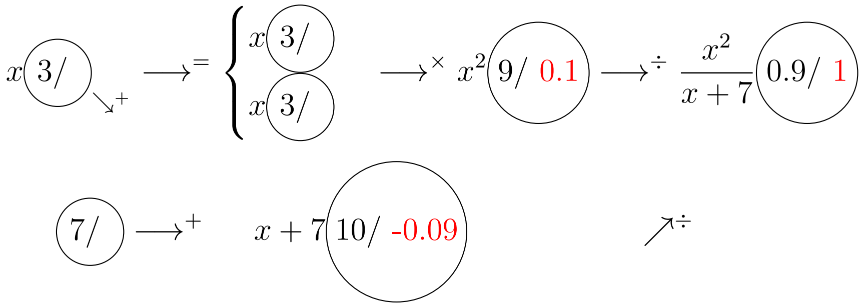
$$\frac{\partial y}{\partial x_k} = 1$$

  Fill in:

$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial x_k}\frac{\partial (x_i/x_j)}{\partial x_i} = \frac{1}{x_j} = \frac{1}{10}$$

  Fill in:

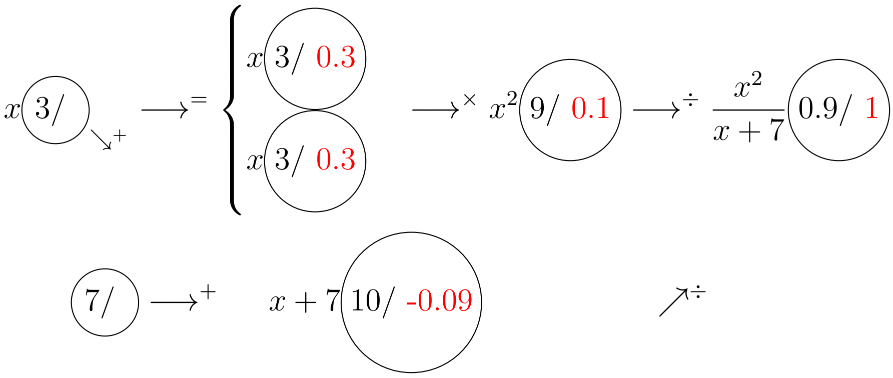
$$\frac{\partial y}{\partial x_j} = \frac{\partial y}{\partial x_k}\frac{\partial (x_i/x_j)}{\partial x_j} = -\frac{x_i}{x_j^2} = -\frac{9}{100}$$



- Second (upper) backpropagation:
  - Let $(x^2) = x_k = x_i x_j = 9$
  - Let $(x) = x_i = 3$, let $(x) = x_j = 3$

- Fill in:

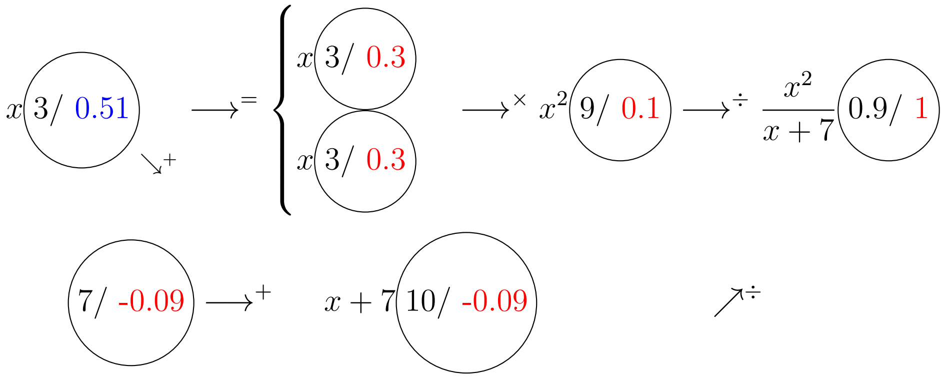
$$\frac{\partial y}{\partial x_i} = \cancelto{0.1}{\frac{\partial y}{\partial x_k}}\frac{\partial x_i x_j}{\partial x_i} = 0.1(3) = {\color{red}0.3} \qquad \text{same for} x_j$$



- Third (lower) backpropagation):
  - Let $(x + 7) = x_k = x_i + x_j = 10$
  - Let $(x) = x_i = 3$, Let $(7) = x_j = 7$
  - Fill in:

$$\frac{\partial y}{\partial x_j} = \cancelto{-0.09}{\frac{\partial y}{\partial x_k}}\frac{\partial (x_i + x_j)}{\partial x_j} = -0.09(1) = {\color{red}-0.09} \tag{123}$$

- Final backpropagation: add objects connecting to original variable ${\color{red}0.3 + 0.3 - 0.09 =}$ ${\color{blue}0.51}$ and this *is* $(\frac{x^2}{x+7})|_{x=3}$.



## 3.5 day22

We have some integral $\int_a^b f = Q(f)$ that we want to approximate with a function $q(y_j) = Q(f)$, where $y_j = f(x_j)$, $x_j$ being distinct points in $[a, b]$. Obviously if we knew the antiderivative $F$ to $f$ such that $F' = f$, then we could do

$$Q(f) = \int_a^b f = F(b) - F(a) = q(y_j),$$

but the antiderivative is often not available.

We let our approximation be of the form

$$q(y_j) = \sum_{j=0}^{m} w_j y_j = (b-a) \sum_{j=1^m} \underbrace{\alpha_j}_{factored\ out\ b-a} y_j, \qquad y_j = f(x_j) \tag{124}$$

We enforce (by definition) $f = 1 \to Q(1) = \int_a^b 1 = (b-a)$. If $m = 3$ (say), then

$$q(y) = (b-a)(\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3) =^{need} b-a \implies \sum_j \alpha_{j=0}^m = 1. \tag{125}$$

We can enforce that polynomials of degree $m^{2m-1 \ in \ hw?}$ must be exact. So if $m = 4$ we can say that $\int_a^b f(x)dx = w_0 f(x_0) + w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3) + w_4 f(x_4)$ should be exact for $f = 1, x, x^2, x^3$, giving us the system of equations of size $2m-1$

$$\begin{cases} \int_a^b 1dx = w_0 + w_1 + w_2 + w_3 + w_4 \\ \int_a^b xdx = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \\ \int_a^b x^2dx = w_0x_0^2 + w_1x_1^2 + w_2x_2^2 + w_3x_3^2 + w_4x_4^2 \\ \int_a^b x^3dx = w_0x_0^3 + w_1x_1^3 + w_2x_2^3 + w_3x_3^3 + w_4x_4^3 \\ \vdots \end{cases} \tag{126}$$

To solve for the unknown weights $w_j$ and nodes $x_j$. Note, symmetry determines that $w_2 = 0, w_1 = w_3, w_0 = w_4$ decreasing the total unknowns/number of needed equations.

The above we can call *Formulation* 1. *Formulation* 2, below, is using Lagrange polynomials to approximate the function:

$$Q(f) = \sum_{j=0}^{m} f(x_j) \underbrace{\int_a^b l_j(x)dx,}_{w_j} \tag{127}$$

where $l_j(x)$ is the $j$th lagrange polynomial with respect to individual node $x_j$. Recall,

$$l_j(x) = \prod_{q=0, q\neq j}^{n} \frac{x - x_q}{x_j - x_q}. \tag{128}$$

The choice of $x_0 = a, x_m = b$ is called the Newton-Cotes rule of degree $m$.

## 3.6 day23

The nodes $x_i$ are also the roots of the $n$th Legendre polynomial. Legendre polynomials $p_0, p_1, \ldots, p_n$, i.e. the polynomials that help obtain the quadrature rule of degree $n$, can be written as

$$p_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2{+}1)^n \qquad p_{0,\dots,4}(x) = \begin{cases} 1 \\ x \\ \frac{1}{2}(3x^2-1) \\ \frac{1}{2}(5x^3-3x) \\ \frac{1}{8}(35x^4-30x^2+3) \\ \vdots \end{cases} , \text{roots} = \begin{cases} 0 \\ \pm 1/\sqrt{3} \\ 0, \pm\sqrt{3/5} \\ \pm\sqrt{3/7 \mp 2\sqrt{6/5}/7} \\ 0, \pm\sqrt{5 \mp 2\sqrt{10/7}}/3 \\ \vdots \end{cases} \tag{129}$$

## 3.7 day24

Solving systems of nonlinear equations

$$F(x) = \begin{Bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{Bmatrix} \underbrace{=}_{choose \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} such\ that} \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{Bmatrix}. \tag{130}$$

For example let's say we want the solution to the circle $v^2 + u^2 = 1$ and the cubic function $v = u^3$. To do this we will find the roots of the system

$$F(x) = \begin{Bmatrix} f(u,v) \\ g(u,v) \end{Bmatrix} = \begin{Bmatrix} u^3 - v \\ u^2 + v^2 - 1 \end{Bmatrix} \underbrace{=}_{choose\ u,v\ such\ that} \{0\} \tag{131}$$

In 1d/newton the core idea is to linearize $F$ at $x_k$ and use the root of the linearization as $x_{k+1}$ ($x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$). If the step length is $s$, then the analog to the linearization in multiple dimensions is

$$F(x) + F'(x)s =^{set} 0 \tag{132}$$

$$\implies \underbrace{F'(x^{(k)})}_{A} s^{(k)} = -F(x^{(k)}) \implies s^{(k)} = -A^{-1}F(x^{(k)}) \qquad x^{(k+1)} = x^{(k)} + s^{(k)}, \tag{133}$$

where the Jacobian is

$$F'(x) = A = f_{i,j} = \frac{\partial f_i}{\partial x_j} \tag{134}$$

## 3.8 day25

$F$ is convex on convex region $D$ if $\forall x, y \in D$,

$$(1-\lambda)F(x) + \lambda F(y) \geq F[(1-\lambda)x + \lambda y], \tag{135}$$

and the region $D$ is convex if $\forall x, y \in D$,

$$(1-\lambda)x + \lambda y \in D, \quad \lambda \in [0,1], \tag{136}$$

i.e. you can draw a straight line between any two points in the region and the entire line also fits in the region. Convexity required for global Newton convergence, but the overall theorem requiring it is very restrictive.

Newton converges "typically" given a good starting guess and does so quadratically (fast). But *not* global, i.e. not any $x^{(0)}$ can be used. This is because if (say) there are two roots you will converge to only one.

To summarize, the Newton method in multi-D is

$$\boxed{x^{(k+1)} = x^{(k)} + s^{(k)}, \qquad F'(x^{(k)})s^{(k)} = -F(x^{(k)})}. \tag{137}$$

That is, do:

Start with $x^{(0)}$

Compute $F(x^{(0)}), A(x^{(0)})$ $\qquad \leftarrow_{\textit{go here}}$

Solve $s^{(0)} = -A^{-1}(x^{(0)})F(x^{(0)})$

Solve $x^{(1)} = x^{(0)} + s^{(0)}$ $\qquad \leftarrow_{\textit{repeat with } x_1}$

Compute $F(x^{(1)}), A(x^{(1)})$, etc.

Quasi Newton methods addresses the fact that calculating $F' = A$ at each iterative step is inefficient by supposing approximations for it. Do:

Start with $x^{(0)}$

Compute $F(x^{(0)}), A(x^{(0)})$, let $B(x^{(0)}) = A(x^{(0)})$

Solve $s^{(0)} = -A^{-1}(x^{(0)})F(x^{(0)})$

Solve $x^{(1)} = x^{(0)} + s^{(0)}$

Compute $F(x^{(1)})$

Solve $\Delta^{(0)} = F(x^{(1)}) - F(x^{(0)})$

BROYDEN UPDATE: $B^{(1)} = B^{(0)} + \frac{\Delta^{(0)} - B^{(0)}s^{(0)}}{s^{(0)T}s^{(0)}}s^{(0)T}$

Solve $s^{(1)} = -B^{-1}(x^{(1)}))F(x^{(1)})$ $\qquad \leftarrow_{\textit{go here}}$

Solve $x^{(2)} = x^{(1)} + s^{(1)}$

Compute $F(x^{(2)})$

Solve $\Delta^{(1)} = F(x^{(2)}) - F(x^{(1)})$

BROYDEN UPDATE: $B^{(2)} = B^{(1)} + \frac{\Delta^{(1)} - B^{(1)}s^{(1)}}{s^{(1)T}s^{(1)}}s^{(1)T}$ $\qquad \leftarrow_{\textit{repeat with } x_2}$ etc.

and with this you only have to do one genuine Jacobian calculation, $A(x^{(0)})$.

## 3.9 day26

We might want to find a local minimum/minimizer of $f$ in an interval. This is done in a way reminiscent of bisection, where we had a bracket $[a, b]$ of a root and the idea was to shrink the bracket repeatedly until tolerance.

For minimization we can do something similar. Let's define "unimodal" by saying $f$ is unimodal on $[a, b]$ if $\exists c \in (a, b)$ such that $f$ is strictly decreasing on $[a, c]$ and strictly increasing on $[c, b]$. Then $c$ is the unique minimizer of $f$ on $[a, b]$.

Let's define a "vee" as a triple $(p, q, r)$ such that $f(p) > f(q) < f(r)$. Then we are guaranteed that $c \in (p, r)$. Similarly to bisection, we want to now shrink the vee. We pick a new point $s \in (p, r)$, find $f(s)$ and update the vee accordingly. (If by accident $f(s) = f(q)$ then pick a new $s$).

If $h$ is the distance between $s$ and $q$ relative to $p$ and $q$ (or $r$ and $q$ depending on the interval $s$ is in), then $h = .618 = \frac{-1+\sqrt{5}}{2}$ guarantees a specific reduction rate of the vee.

Note, for quadratic minima/minimizers corresponding to quadratics, the vee near $c$ hardly changes at all as $x$ changes, meaning such a strong convergence tolerance as $10^{-15}$ is unrealistic; something like $10^{-8}$ is way more common even though it is huge compared to $\epsilon_{mach}$.

## 3.10 day27

Minimization of multivariable $f(x_1, x_2, \dots)$ extends from the scalar case discussed previously. In scalar case we look for $f' = 0$, and in multi-D case we are looking for $\nabla f = 0$. Letting $g = \nabla f$,

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)} \tag{138}$$

n-D:

$$x^{(k+1)} = x^{(k)} - g'(x^{(k)})^{-1} g(x^{(k)}) \tag{139}$$

where

$$g(x) = \begin{Bmatrix} f_{,1}(x) \\ f_{,2}(x) \\ \vdots \\ f_{,n}(x) \end{Bmatrix} \tag{140}$$

and the Hessian matrix

$$g'(x) = \begin{bmatrix} f_{,11}(x) & f_{,12}(x) & f_{,13}(x) & \dots & f_{,1n}(x) \\ f_{,21}(x) & f_{,22}(x) & f_{,23}(x) & \dots & f_{,2n}(x) \\ \vdots & & & & \\ f_{,n1}(x) & f_{,n2}(x) & f_{,n3}(x) & \dots & f_{,nn}(x) \end{bmatrix} \tag{141}$$

Gradient descent:

at $x^{(k)}$, the direction in which $f$ decreases the fastest is $-\nabla f(x^{(k)})$. Note, $\nabla f$ is orthogonal/perp to level curves/contours of $f$. Therefore, we can define the interval as the line spanned by $-\nabla f(x_k)$, find the minimum on that line, and do the same thing for

$-\nabla f(x_{k+1})$, etc. A crude way to ”constantly” follow the local negative gradient is to use the iteration

$$x^{(k+1)} = x^{(k)} - h\nabla f(x^{(k)}), \quad h\ small. \tag{142}$$

Conjugate gradient descent is best shown through example. Let

$$f(x) = 2 + 4x + 7y + 5x^2 + 13y^2 + 6xy \tag{143}$$

$$\implies f(x) = 2 - \begin{Bmatrix}-4 & -7\end{Bmatrix}\begin{Bmatrix}x \\ y\end{Bmatrix} + \begin{Bmatrix}x & y\end{Bmatrix}\begin{bmatrix}5 & 3 \\ 3 & 13\end{bmatrix}\begin{Bmatrix}x \\ y\end{Bmatrix} \tag{144}$$

$$\implies f(x) = c - b^T x + x^T A x. \tag{145}$$

Do:

```
• Residual $r_0 = b - Ax_0$
• $p_0 = r_0$
• $k = 0$
• Repeat:
      Rayleigh $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
      $x_{k+1} = x_k + \alpha_k p_k$
      $r_{k+1} = r_k - \alpha_k A p_k$
      if $r_{k+1}$ sufficiently small then exit loop
      $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
      $p_{k+1} = r_{k+1} + \beta_k p_k$
      $k = k + 1$
• Return $x_{k+1}$ as result
```