Numerical Analysis
Math 128A, Summer 2019
Lecture 1 6/24/2019

Remark: This set of notes is made public in hopes of being helpful in consolidating understanding. It is unofficial and guaranteed to contain errors. If you spot any such inaccuracies or have any comments, please let me know at dsuryakusuma@berkeley.edu.

Lecturer: Professor John A. Strain
`https://math.berkeley.edu/~strain/128a.m19/index.html`

Class: M/Tu/W/Th 11-1pm (289 Cory)
OH: M,Tu 1-2pm (891 Evans)

Textbook: Burden & Faires (BFB) (req'd); Gander, Gander, & Kwok (GGK)

Grading:
- Homework: 30% ; Due weekly on Wednesdays
- MT: 30% ; Wed July 17 ; 11am - 1pm (289 Cory)
- Final: 40% ; Thurs Aug 15 ; 11am - 1pm (289 Cory)

$\geq$ Matlab required (or something better than Matlab)
- vector, matrix, for-loop, functions, plotting

1 Review of Calculus

Theorem 1.1. Intermediate Value Theorem -

For continuous function $f : [a, b] \to \mathbb{R}$,
for any y, there exists $a \leq b, \quad f(c) = y$.

$$\min\{f(a), f(b)\} \leq y \leq \max\{f(a), f(b)\}$$

Example:
For the first few lectures, we'll be looking at a two-step process:
(1) Solve $f(x) = 0$ by bracketing $a \leq x \leq b$ where:

$$f(a) \leq 0 \leq f(b) \text{ or } f(b) \leq 0 \leq f(a),$$

i.e. sign $f(a) \leq$ sign $f(b)$. For example, consider:

$$\ln x = \begin{cases} 1 & x = e \\ -1 & x = 1/e \end{cases}$$

Then there exists $\frac{1}{e} = \frac{1}{2.72} < x < e = 2.72$. So $x \ln x = 0$.
(2) Shrink bracket somehow, with the goal of getting a better solution of $x \ln x = 0$.

Theorem 1.2. Fundamental Theorem of Calculus (FTC)
For f, f' continuous, we have:

$$f(t) = f(a) + \int_a^t f'(s)\ ds$$

Note: We use s here to not get confused with x or t.

Now we derive the Mean Value Theorem (and "everything else we're going to need") using the FTC (hence 'fundamental theorem of calculus').

Theorem 1.3. Mean Value Theorem:
Averaging

$$\text{IVT: } \underbrace{\min_{a\le x\le b} f(x)}_{\text{constant}} \le f(s) \le \underbrace{\max_{a\le x\le b} f(x)}_{\text{constant}}$$

$$0 < a(x) \le b(x)$$
$$0 < c(x) \le d(x)$$

$$a \le b \implies -a \ge -b \implies -b \le -a$$

Example:

$$\int_a^b \left[\min_{a\le x\le b} f(x)\right]\ dx \le \int_a^b f(s)\ ds \le (b-a)\max_{a\le x\le b} f(x)$$

$$\min_x f(x) \le \frac{1}{b-a}\int_a^b f(s)\ ds = \overline{f} = \overline{y} \le \max_x f(x)$$

So there is some number c with $a \le c \le b$ such that

$$f(c) = \overline{f} = \frac{1}{b-a}\int_a^b f(s)\ ds.$$

MVT for Integrals:

$$\int_a^b f(s)\ ds = (b-a)f(c)$$

for some $c \in [a, b]$, c is unknown, but usually enough to know c exists.

Example: $a \le b \to ga \le gb$ (if $g > 0$)

$$g(s)[\min f(x)] \le f(s) \le g(s)[\max f(x)]$$

$$\int_a^b g(s)[\min f(x)] \le \int_a^b f(s) \le \int_a^b g(s)[\max f(x)]$$

$$\min f(x) \le \frac{\int_a^b g(s)f(s)\ ds}{\int_a^b g(s)\ ds} \le \max f(x)$$

Then the 'expectation' of f is $\overline{f}$ or $\langle f \rangle$ or $E(f)$.

MVT for Integrals:
The IVT tells us the same thing: $\exists c \in [a, b]$ such that

$$\int_a^b g(s)f(s)\ ds = f(c) \int_a^b g(s)\ ds$$

$$\begin{aligned}\frac{f(x) - f(a)}{x - a} &= \frac{1}{x - a}\int_a^x f'(s)\ ds \\ &= f'(c) \quad \text{for some unknown } c \text{ between } a \text{ and } x.\end{aligned}$$

Another way of saying this is:

$$f(x) - f(a) = f'(c)(x - a)$$

If you know the derivative of a function, then you can control how far apart the values are.

Example: $f(x) = \sin(x) \qquad f'(x) = \cos(x)$
$\sin(x) - \sin(a) = \cos(c)(x - a)$
So if x is close to a, then cos is always less than or equal to 1.

$$|\sin(x) - \sin(a)| = |\cos(c)(x - a)| \leq |x - a|$$

So sin is like a contraction, where mistakes going through a process will be 'redeemed' and 'forgotten', as long as our expressions have bounded derivatives.

Theorem 1.4. Taylor's Theorem (generalization of MVT)

$$\begin{aligned}f(x) &= f(a) + \int_a^x [1]f'(s)\ ds \\ &\qquad \text{(use int. by parts ; integrate the 1, differentiate the } d'(s)) \\ &= f(a) - \int_a^x \left[\frac{d}{ds}(x - s)\right] f'(s)\ ds \\ &= f(a) - (x - s)f'(s)\Big|_a^x - \left(\int_a^x -(x - s)f''(s)\ ds\right) \\ &= f(a) + (x - a)f'(a) + \int_a^x (x - s)f''(s)\ ds \\ &= f(a) + (x - a)f'(a) + \int_a^x \left[-\frac{d}{ds} \cdot \frac{(x - s)^1}{1!}\right] f''(s)\ ds \\ &= \frac{(x - a)^0}{0!}f(a) + \frac{(x - a)^1}{1!}f'(a) + \frac{(x - a)^2}{2!}f''(a) + \cdots\end{aligned}$$

Common Taylor Expansions:

$$\begin{aligned} f(x) = e^{bx} \quad f'(x) = be^{bx} = f^{(k)}(x) = b^k e^{bx} \\ f(0) = 1 \quad f'(0) = bf^{(k)}(0) = b^k \\ e^{bx} = \frac{x^0}{01}1 + \frac{x^1}{1!}b + \frac{x^2}{2!}b^2 + \frac{x^3}{3!}b^3 + \cdots \\ e^{bx} = \sum_{k=0}^{\infty} \frac{b^k}{k!}x^k \end{aligned}$$

Writing infinite series like that something $= \cdots$ means that the remainder is

$$\sum_{k=K}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k = R_K(x) \to 0_{\text{as } K \text{ goes to } \infty}$$

Also $f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \cdots$, let's say we stop at some

$$\frac{(x-a)^{k-1}}{(k-1)!}f^{(K-1)}(a) + \underbrace{\int_a^x \frac{(x-s)^{k-1}}{(k-1)!}f^{(k)}(s)\ ds}_{R_K(x)}$$

with $f(s) > 0$. So we have, for the error of a Taylor Polynomial stopping BEFORE the k-th term:

$$\begin{aligned} R_k(x) &= f^{(k)}(c)\int_a^x \frac{(x-s)^{(k-1)}}{(k-1)!}\ ds \\ &= f^{(k)}(c)\left[\frac{-(x-s)^k}{k!}\right]_a^x \\ &= f^{(k)}(c)\frac{(x-a)^k}{k!} \end{aligned}$$

Hence, for our example b^{bx} above:

$$|R_k(x)| \le b^k e^{bx}\frac{(x-a)^k}{k!}$$

Stirling's Approximation:

$$k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k$$

Example:

$$\begin{aligned}
f(x) &= \frac{1}{1-x} = \sum_{k=0}^{\infty} x^k \quad \text{, if } |x| < 1, \quad 0 < c < x \\
f'(x) &= \left((1-x)^{-1}\right)' = 1!(1-x)^{-2} \\
f'(0) &= 1! \\
f''(x) &= 2!(1-x)^{-3} \\
f''(0) &= 2!
\end{aligned}$$

So we check, with our formula

$$\begin{aligned}
R_k(x) &= f^{(k)}(c)\frac{(x-0)^k}{k!} = k!(1-c) \\
&= k!(1-c)^{-k-1}\frac{x^k}{k!} \\
&= \frac{1}{1-c}\left(\frac{1}{1-c}\right)^k \\
&\leq \frac{1}{1-x}\left(\frac{x}{1-x}\right)^k \\
&\leq \frac{2^{-k}}{1-x} \text{ if } x \leq \frac{1}{3}
\end{aligned}$$

So we have a bound for the relative error. Question: How big does k have to be until we don't care anymore? 10 is usually enough for mechanical engineering. 6-digit accuracy can be good enough for electric engineering. For bio-engineering and heart surgery, we'll probably want 9-digit accuracy.

Definition: Taylor Polynomial, Remainder (BFB) -

Let $P_n(x)$ be the nth **Taylor polynomial** for f about x_0, and $R_n(x)$ be the **remainder term** or **truncation error** associated with $P_n(x)$. Suppose $f \in C^n[a,b], f^{(n+1)}$ exists on $[a,b]$, and $x_0 \in [a,b]$. For every $x \in [a,b]$, there exists a number $\xi(x)$ between x_0 and x with:

$$f(x) = P_n(x) + R_n(x),$$

$$\begin{aligned}
P_n(x) &= \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k \\
R_n(x) &= \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1}
\end{aligned}$$

Alternative to Taylor expansion:

$$
\begin{aligned}
f(1) &= f(0) + \int_0^1 \underbrace{1}_{-\frac{d}{ds}(1-s)} f'(s)\, ds \\
&= f(0) + \int_0^1 \underbrace{1}_{-\frac{d}{ds}(\frac{1}{2}-s)} f'(s)\, ds \\
&= f(0) + \underbrace{\left(-(\frac{1}{2}f'(s)\big|_0^1)\right)}_{-\frac{1}{2}f'(1)+\frac{1}{2}f'(0)} - \int_0^1 - \left[\frac{1}{2} - s\right] f''(s)\, ds \\
&= f(0) - \frac{1}{2}f'(1) + \frac{1}{2}f'(0) - \int_0^1 - \left[-\frac{d}{ds}\frac{(\frac{1}{2}-s)^2}{2!}\right] f''(s)\, ds \\
&= f(0) - \frac{1}{2}\left[f'(1) - f'(0)\right] - \frac{1}{2!}\left[\left(\frac{1}{2} - s\right)^2 f''(s)\right]_0^1 \\
&\qquad + \int_0^1 -\frac{d}{ds}\frac{(\frac{1}{2}-s)^3}{3!} f''(s)\, ds \\
\implies f(1) - f(0) &\approx -\frac{1}{2}\left[f'(1) - f'(0)\right] - \frac{1}{8}\left[f''(1) - f''(0)\right]
\end{aligned}
$$

2 Computer Arithmetic

Floating Point arithmetic $\approx$ real number arithmetic

64-bit IEEE Standard Arithmetic:

$$\underbrace{(-1)^s}_{\text{(sign, 1 bit)}} \quad \underbrace{2^{c-1023}}_{\text{(Characteristic, 11-bit exponent)}} \quad (1 + \quad \underbrace{f}_{\text{(mantissa, 52-bit binary fraction)}} \quad)$$

1023 is the bias applied to c.

$$[(-1)^s]\ [2^{e-1023}]\ [\ \underbrace{1.}_{\text{implied}}\ b_1 \dots b_{52}]$$

So the mantissa gives the numbers between 1 and 2 (not 0 1).

Definition: machine epsilon -

$$\varepsilon := 2^{-52}$$

$$\begin{gathered} \dots \quad |, \\ \frac{1}{2} - \frac{\varepsilon}{4} \quad |, \\ \frac{1}{2}, \dots, 1 - \frac{3\varepsilon}{2}, 1 - \frac{\varepsilon}{2} \quad |, \\ 1, 1+\varepsilon, \dots, 2-\varepsilon \quad |, \\ 2, 2+2\varepsilon, \dots, 4-2\varepsilon \quad |, \\ 4, 4+4\varepsilon, \dots \end{gathered}$$

So there are 2^{11} logarithmic intervals $[2^{+(k-1)}, 2^{+k}]$. On the interval $[2^{k-1}, 2^k)$, there are 2^{52} floating point numbers ("FP#s").

Example: Consider the number:

$$0 \quad 10000000011 \quad 1011100100010 \cdots 0$$

First bit is 0, so $s = 0$ and the number is positive.
The next 11 bits gives the characteristic, equal to:

$$c = 1 \times 2^{10} + 1 \times 2^1 + 1 \times 2^0 = 1024 + 2 + 1 = 1027$$

So the exponent part is $2^{1027-1023} = 2^4 = 16$.
The last 52 bits specify that the mantissa is

$$f = 1\left[\frac{1}{2}\right]^1 + 1\left[\frac{1}{2}\right]^3 + 1\left[\frac{1}{2}\right]^4 + 1\left[\frac{1}{2}\right]^5 + 1\left[\frac{1}{2}\right]^8 + 1\left[\frac{1}{2}\right]^{12}.$$

So in total, the machine number equals:

$$\begin{aligned} (-1)^s 2^{c-1023}(1+f) &= (-1)^0 2^{1027-1023}\left[1 + \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right)\right] \\ &= 1 \times 16 \times (\cdots) \\ &= 27.56640625. \end{aligned}$$

Definition: Rounding -

$$fl(x) := \text{ nearest FP \# to } x$$

$$fl : \mathbb{R} \to FP\#S$$

$$2.718 \to 2.72$$

Round to "even" (last digit 0) if there is a tie. For example,

Example of tie: How do we round $1.111 \to ?$
$2 - \frac{1}{8} = 10.0$ last bit 0, is the stable choice
As opposed to $2 - \frac{1}{4} = 1.11$, which we don't choose.

Remark: END LECTURE 1 HERE

Math 128A, Summer 2019
Lecture 2, Tuesday 6/25/2019

1 Review of Lec 1 Floating Point Arithmetic

In last lecture, we ended with rounding FPN (Floating Point Numbers), and tie-breaking.

Definition: Normal number -

$$[S] \quad [C] \quad [f]$$

$$(-1)^s \cdot 2^{(C-1023)} \cdot (1+f)$$

(65-bit effective by adding the implied 1 in mantissa)

Definition: Subnormal number -

$$C := 0$$

$$\implies (-1)^S \cdot 2^{-1022} \cdot (f)$$

"Gradual underflow". We lose the $1 + \cdots$, but we get another 51 bits before our numbers go to zero $2^{-1023} \to 2^{-1022}$.

Note: We used to use "fixed-point" arithmetic (with all non-sign bits assigned to the mantissa; i.e. no characteristic value) back when computers were vacuum tubes in a room. Now with the characteristic, we get better usability.

Other special cases:

Consider that ± 0 are both "standard numbers". "$\pm$Inf" as well. But there's NaN (not a number), which is anything that has no valid representation.

1.1 Spacing of FPNs

Definition: Machine Epsilon (mach eps) -
Also called **ULP** for "unit in the last place".

$$\varepsilon := 2^{-52}$$

So we have the following spacing:

$$1, 1+\varepsilon, 1+2\varepsilon, \ldots, 2-\varepsilon, 2, 2+2\varepsilon, 2+4\varepsilon, \ldots, 4-2\varepsilon, 4, \cdots$$

Example: Constructing the IEEE FP representations:

$$\begin{aligned}
1 = S = 0, C = 01111111111 = 1023, f = 0 \cdots 00 \\
2 = S = 0, C = 10000000000 = 1024, f = 0 \cdots 00 \\
2 + 2\varepsilon = S = 0, C = 10000000000 = 1024, f = 0 \cdots 01 \\
2 + 4\varepsilon = S = 0, C = 10000000000 = 1024, f = 0 \cdots 10 \\
2 + 6\varepsilon = S = 0, C = 10000000000 = 1024, f = 0 \cdots 11 \\
1 - \frac{\varepsilon}{2} = S = 0, C = 1022, f = 1 \cdots 1
\end{aligned}$$

Definition: Rounding -

Rounding: $fl : \mathbb{R} \to$ nearest FPN

Example:

$$\text{fl}\ (1 + \varepsilon + \frac{\varepsilon}{\pi}) = 1 + \varepsilon \text{fl}\ (1 - \frac{\varepsilon}{\pi})$$

Resolving ties: (to last bit zero)

$$\begin{aligned}
&fl\left(1 + \frac{\varepsilon}{2}\right) = 1 \\
&fl\left(1 - \frac{\varepsilon}{2}\right) = 1 - \frac{\varepsilon}{2} \\
&fl\left(1 - \frac{\varepsilon}{4}\right) = 1 \quad \text{(this is more interesting)}
\end{aligned}$$

Note that a real number will live in between two FPNs, one will be even and one will be odd. It's not necessarily rounding down; just rounding to 0. This prevents bias accumulating ("drift") over many operations.

Remark: Rounding is "monotone", which means it preserves order (inequalities).

$$a < b \implies fl(a) \leq fl(b)$$

Note the possible equality if they both round to the same FPN.

Aside/Motivation: debugging a program to run on supercomputer

$$\arccos(x); x = \frac{a}{a^2 + b^2}$$

But now with the IEEE standard, we don't have these issues with inequalities, where values go out of 'bounds'.

2 Rounding Error

$$\begin{aligned}
1 \leq x < 2 : \qquad |fl(x) - x| &\leq \varepsilon \\
&\leq \frac{\varepsilon}{2} \\
&\leq \frac{\varepsilon}{2}|x| \\
\text{Relative Error} =: \quad \frac{|fl(x) - x|}{|x|} &\leq \frac{\varepsilon}{2} \leq \varepsilon
\end{aligned}$$

But of course, if we have a large x, our precision is much smaller because we need to use bits to store the size.

If we have the following addition:

$$\begin{aligned} &0.00\overbrace{1\cdots1}^{52} \\ &+1.000\cdots0 \\ &=1.\underbrace{001\cdots1|11}_{54} = 1.010\cdots0 \\ &\quad = [2^{-3}\cdot(1-\varepsilon)] + [1] = \underbrace{1+2^{-3}-2^{-3}\varepsilon}_{\text{exactly}} \\ fl(x,y) &= fl(1+2^{-3}-2^{-3}\varepsilon) = 1+2^{-3} \end{aligned}$$

Mantra of FP Arithmetic: Let x, y be FPN, floating point numbers. The floating point result of any binary operation on two FPNs gives us:

$$fl(x+y) = \text{the exact result, correctly rounded}$$

And likewise for $fl(xy), fl(x-y), fl(x/y), fl(\sqrt{x})$.

So instead of doing FP arithmetic per term, we just perform the regular math, then round at the end.

Example: $0.00\underbrace{11\cdots11}_{52} = 2^{-3}\times 1.\underbrace{\overbrace{1\cdots1}^{f}}_{2-\varepsilon}$

Or equivalently, $fl(1+2^{-3}(2-\varepsilon)) = fl(1+2^{-2}-2^{-3}\varepsilon) = 1.01$.

So we have:

Definition: Relative Error $|\delta| \le \frac{\varepsilon}{2}$ -

$$\begin{aligned} \frac{|fl(x)-x|}{|x|} &\le \frac{\varepsilon}{2} \\ |\underbrace{fl(x)-x}_{\delta x}| &\le \frac{\varepsilon}{2}|x| \\ fl(x)-x &= \delta x \\ fl(x) &= x+\delta x \\ &= x(1+\delta), \quad |\delta| \le \frac{\varepsilon}{2} \end{aligned}$$

Remark: δ is INDEPENDENT of x.

Thus:

$$\begin{aligned}
fl(x \pm y) &= (x \pm y)(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl(\sqrt{x}) &= \sqrt{x}(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2} \\
fl((x+y)+z) &\neq (x+y+z)(1+\delta) \\
&= fl[fl(x+y)+z] \\
&= fl[(x+y)(1+\delta)+z] \\
&= [(x+y)(1+\delta_1)+z](1+\delta_2) \\
&= (x+y)(1+\delta_1)(1+\delta_2)+z(1+\delta_2) \\
&= (x+y)(1+\delta_1+\delta_2+\delta_1\delta_2)+z(1+\delta_2) \\
&= (x+y)(1+2\delta_3)+z(1+\delta_2), \qquad |\delta_3| \leq \frac{3}{2} \\
&= x(1+2\delta_3)+y(1+2\delta_3)+z(1+\delta_2)
\end{aligned}$$

Note that these deltas can be different. But each of these terms are very close to their respective true values x, y, z.

Definition: Backwards Error Analysis -
Take $fl(x+y+z) = \hat{x}+\hat{y}+\hat{z}$ where:

$$\begin{aligned}
\hat{x} &:= x(1+2\delta_3) \\
\hat{y} &:= y(1+2\delta_3) \\
\hat{z} &:= z(1+\delta_2)
\end{aligned}$$

As opposed to...

Definition: Forward Error Analysis -

$$\begin{aligned}
\frac{|fl(x+y+z)-(x+y+z)|}{|x+y+z|} &\leq \frac{|(x+y)(1+2\delta_3)+z(1+\delta_2)-(x+y+z)|}{|x+y+z|} \\
&\leq \frac{2|\delta_3||x+y|+|z||\delta_2|}{|x+y+z|} \\
&\leq \frac{2|x+y|+|z|}{|x+y+z|}\frac{\varepsilon}{2}
\end{aligned}$$

Example: Take $x = 1+\varepsilon, y = -\varepsilon, z = -1$. Recall that we had $1 \leq x < 2$, with $|fl(x)-x| \leq \frac{\varepsilon}{2}|x|$.

Our relative error bound is

$$\frac{2|x+y|+|z|}{|x+y+z|}\frac{\varepsilon}{2} \leq \frac{3}{0}\cdot\frac{\varepsilon}{2}$$

But we have division by zero, which is useless, so our result would be unverifiable.

Mantra 2 of Computing: Don't compute 0 by subtracting nonzero objects (because then we can't tell the relative error).

Key Takeaway:

$$fl(x) = x(1+\delta), \qquad |\delta| \leq \frac{\varepsilon}{2}$$

Example of large number of operations, backwards error analysis:

Suppose we want to find:

$$S_n = \sum_{j=1}^{n} x_j$$

We need an algorithm:

$$S_1 = x_1 \tag{1}$$
$$S_2 = S_1 + x_2 \tag{2}$$
$$\vdots \tag{3}$$
$$S_n = S_{n-1} + x_n \tag{4}$$

Pseudocode

```
S_n = S_{n-1} + x_n
unless
n = 1    when S_1 = x_1
recursion
```

In numerical analysis we tend to avoid recursion because it can be expensive (number of operations).

We conclude:

$$\begin{aligned} fl(S_n) = x_1[1+(n-1)\delta_1] & \\ + x_2[1+(n-1)\delta_2] & \\ + x_3[1+(n-2)\delta_3] & \\ + \cdots + x_n[1+\delta_n] & \end{aligned}$$

or more sloppily (overestimating),

$$\begin{aligned} fl(S_n) = x_1[1+(n)\delta_1] & \\ + x_2[1+(n-1)\delta_2] & \\ + x_3[1+(n-2)\delta_3] & \\ + \cdots + x_n[1+\delta_n] & \end{aligned}$$

So BEA (backwards error analysis) says that x has n errors.

If we want FEA (forward error analysis) and its diagnostic, then we look at:

$$\left|\frac{fl(S_n) - S_n}{S_n}\right| \leq \frac{n|x_1| + (n-1)|x_2| + \cdots + 1|x_n|}{|S_n|} \cdot \frac{\varepsilon}{2}$$

Remark: Notes end here for today.

Big ideas: BEA, FEA, $(1+\delta_n)$ and simplifications for compounding error from a sequence of operations.

Math 128A, Summer 2019

Lecture 3, Wednesday 6/26/2019

CLASS ANNOUNCEMENTS:
Instructions/Tips for problem sets,

- different page for each problem
- don't try to save paper, (write large)
- don't type (unless you check it a lot)
- include code and output plots
- credit your sources
- don't use proof by contradiction, because it's easy in numerical analysis and **constructive mathematics**, it's easy to make mistakes and hard to find them; additionally, proofs by contradiction *rarely illuminate* the underlying ideas
- be self-critical (review your work; the best growth in Math is when you find your own mistakes)

Today's topics:

- Floating Point
- Algorithms
- big-O notation

1 Floating Point Arithmetic

Recall the Mantra: "Binary Floating Point operations $\{+, -, \times, /, \sqrt{}\}$ on Floating Point numbers deliver **the exact result, correctly rounded**."

Quantitatively, this is equivalent to (or implies) that

$$\begin{aligned} fl(x+y) &= (x+y)(1+\delta), & |\delta| &\leq \frac{\varepsilon}{2} \\ fl(x-y) &= (x-y)(1+\delta), & |\delta| &\leq \frac{\varepsilon}{2} \\ fl(x\times y) &= (x\times y)(1+\delta), & |\delta| &\leq \frac{\varepsilon}{2} \\ fl(x/y) &= (x/y)(1+\delta), & |\delta| &\leq \frac{\varepsilon}{2} \\ fl(\sqrt{x}) &= (\sqrt{x})(1+\delta), & |\delta| &\leq \frac{\varepsilon}{2} \end{aligned}$$

Remark: Floating point arithmetic is **NOT** associative.

That is,

$$fl[\underbrace{(x+y)}_{\text{this error gets compounded}} + z] \neq fl[x + (y+z)]$$

Rule 1: Don't compute 0.

Rule 2: Minimize (the values of) intermediate results if possible.

2 Algorithms

Example: Compute the sum of n numbers: S_n.

```
function [S] = sumn(X,n)

% X is array of values
% n is how many terms we want to add

S = 0;
for j = 1:n
  S = S + X(j);
end

end
```

How much did this algorithm cost? Cost of `sumn(X,n)` is n adds, or "flops".

Definition: Big-O -

We say $f(n) = O(g(n))$ to be equivalent to $\exists K, N$ with

$$|f(n)| \leq Kg(n)$$

for $n \geq N$.

Example:

$$\begin{aligned} 17n^2 &= O(n^2 \log n) \\ n^2 &= O(2^n) \\ 10^{17}n^3 &= O(1.0000001^n) \end{aligned}$$

Aside:

$$\begin{aligned} O(n) &= O(n \log n) \\ &= O(n|\log \varepsilon|) \end{aligned}$$

The floating point result for calulating

$$fl(S_n) = fl[\underbrace{fl(S_{n-1})}+X_n]$$

floating point result of this algorithm is well defined

$$\begin{aligned} &= [fl(S_{n-1}) + X_n](1+\delta_n), \qquad |\delta_n| \leq \frac{\varepsilon}{2} \\ fl(S_n) - S_n &= fl[S_{n-1}](1+\delta_n) - S_n + x_n(1+\delta_n) \\ &= fl(S_{n-1})(1+\delta_n) - S_{n-1}(1+\delta_n) + S_{n-1}\delta_n + X_n\delta_n \\ &= fl[S_{n-1}](1+\delta_n) - S_{n-1}(1+\delta_n) + S_n\delta_n \\ &= [fl(S_{n-1}) - S_{n-1}](1+\delta_n) + \delta_n S_n \\ E_n &= E_{n-1}(1+\delta_n) + \delta_n S_n \end{aligned}$$

where δ_n is a nice error, relative to what we are trying to compute.

Then,

$$
\begin{aligned}
|E_n| &\le |E_{n-1}|\left(1+\frac{\varepsilon}{2}\right)+\frac{\varepsilon}{2}|S_n| \\
|E_{n-1}| &\le |E_{n-2}|(1+\tfrac{\varepsilon}{2})+\frac{\varepsilon}{2}|S_{n-1}| \\
|E_n| &\le |E_{n-2}|\left(1+\frac{\varepsilon}{2}\right)^2+\frac{\varepsilon}{2}\left(1+\frac{\varepsilon}{2}\right)|S_{n-1}|+\frac{\varepsilon}{2}|S_n| \\
&\le \ddots \\
&\le |E_1|\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-1}}+\frac{\varepsilon}{2}\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-2}}|S_2|+\frac{\varepsilon}{2}\underbrace{\left(1+\frac{\varepsilon}{2}\right)^{n-3}}|S_3|+\cdots+\frac{\varepsilon}{2}|S_n| \\
&\quad \text{these act like } O(n\varepsilon^2) \\
&\le \frac{\varepsilon}{2}\left(|S_1|+|S_2|+\cdots+|S_n|\right)+O(n\varepsilon^2)
\end{aligned}
$$

We assert (as it's true) that we compute S_2, then S_3, etc, and calculate and store along the way.

Algorithm for e^x
Let $e^x := e^{m+r} = [e^m][e^r]$, where $m \in \mathbb{Z}$ and $r = x - m$.

Range reduction:

1. store e
2. compute e^m; take m, turn it into binary and

$$m = \sum_{k=0}^{N} b_k 2^k$$

and

$$e^{2^k} = \left(e^{2^{k-1}}\right)^2$$

is in $\log m$ time.

```
function y = expx(x)
% x is the exponent of e
% we won't take a tolerance for precision for now

  m = round(x) % nearest integer to x
```

$$|r| \le 1/2, \quad k! \approx \left(\tfrac{k}{e}\right)^k \implies \tfrac{r^k}{k!} \approx \left(\tfrac{e}{2k}\right)^k = 10^{-15}$$

$$k \ge 15$$

should be enough

$$
\begin{aligned}
|r| = |x-m| &\le 1/2 \\
e^r &= \sum_{k=0}^{q}\frac{r^k}{k!}+O\left(\frac{e}{2q}\right)^q \\
e^r &= 1+\frac{1}{1!}+\frac{r^2}{2!}+\frac{r^3}{3!}+\cdots
\end{aligned}
$$

Remark: When summing a Taylor Series, we typically sum from right to left, for smaller values first, storing, then adding bigger terms as we go.

Definition: Horner's Rule -

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

We compute in $\rightarrow$ out.

```
oursum = 1 / (q!)
for i = (q-1):(-1):0
  oursum = 1 / (i!) + r * oursum
end

y = oursum * (e^m)
```

Example: Computing this is difficult:

$$\frac{e^x - 1}{x}$$

as the numerator nears 0 for small x, and so does the denominator

3 Convergence, Characterizing Error

Definition: Rate of Convergence -

An algorithm producing $p_n \approx p$. If

$$\left|\frac{p_n - p}{p}\right| \leq O\left(n^{-r}]\right)$$

The rate of convergence is $O(n^{-r})$, for example "second-order" $r = 2$, $O(n^2)$.

E.g.

$$e^r - \sum_{k=0}^{q} \frac{r^k}{k!} = O\left[\left(\frac{e}{2q}\right)^q\right]$$

, where big-O is the rate of convergence.

Example: Approximating a derivative. Define:

$$D_h f(x) = \frac{f(x+h) - f(x)}{h} = O(h^{\text{what order?}})$$

(1) one way is MVT:

$$f(x+h) - f(x) = f'(c)h, \qquad \text{for some } x \leq c \leq x+h$$

MVT is a bit too crude here, so we probably want to use one more term in the Taylor expansion.

(1, modified)

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(c)h^2$$

Then we have

$$D_h f(x) := f'(x) + \frac{1}{2}f''(c)h$$

So the Rate of Convergence (RoC) is $O(h^1) = O(1)$, and we call the "Order of convergence" 1.

Remark: Ok, great, now we have some idea of algorithms and cost. What happens in floating-point arithmetic? In the above section, we assume that our approximations are exact.

4 Rate of Convergence in Floating Point Arithmetic

Example:

$$\begin{aligned}
fl[f(x)] &= f(x)(1+\delta_1) \\
fl[f(x+h)] &= f(x+h)(1+\delta_2) \\
fl[F_h f(x)] &= \frac{[f(x+h)(1+\delta_2) - f(x)(1+\delta_1)]\,(1+\delta_3)}{h}(1+\delta_4) \\
&= \frac{f(x+h) - f(x)}{h} + \frac{f(x+h)(3\delta_5)}{h} + \frac{f(x)(3\delta_6)}{h} \\
&= f'(x) + O(h) + O\left(\frac{\varepsilon}{h}\right)
\end{aligned}$$

Don't let $h \to 0$, due to the last term. How do we deal with this? There's a sweet spot in the middle somewhere, where $O(h)$ doesn't explode and $O(\frac{\varepsilon}{h})$ doesn't explode. To find this, we suppose:

$$O(h) + O(\frac{\varepsilon}{h}) := O(1) + O(\frac{-\varepsilon}{h^2})$$

where $h^2 \approx \varepsilon \implies h \approx \sqrt{\varepsilon}$.

So the best achievable error is about:

$$= O(\sqrt{\varepsilon}) + O\left(\frac{\varepsilon}{\sqrt{\varepsilon}}\right) = O(\sqrt{\varepsilon}) = O(10^{-7})$$

Centered 2nd-order approximation: If

$$D_h^2 f(x) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2)$$

Then $\min O(h^2) + O(\frac{\varepsilon}{h}) \implies h = \frac{\varepsilon}{h^2} \implies h^3 = \varepsilon \implies h = \varepsilon^{1/3}$.

So the best achievable error is at

$$O(\varepsilon^{2/3}) + O(\frac{\varepsilon}{\varepsilon^{1/3}}) = O(\varepsilon^{2/3}) = O(10^{-10})$$

Remark: Today's lecture ends today. Tomorrow we'll talk about bisection.

5 Notes from Textbook

Definition: linear, exponential (growth of error) -

- If $E_n \approx CnE_0$, where C is a constant independent of n, then the growth of error is said to be **linear** and **stable**.
- If $E_n \approx C^nE_0$, where $C > 1$, then the growth of error is called **exponential** and **unstable**.

Remark: Linear growth of error is usually unavoidable, and results are generally acceptable when C and E_0 are small.

Definition: Rate (Order) of Convergence -

Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero and $\{\alpha\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with

$$|\alpha_n - \alpha| \leq K|\beta_n|, \quad \text{for large } n,$$

then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with **rate/order of convergence** $O(\beta_n)$. We read this "Big Oh of Beta N". In text, we write it as $\alpha_n = \alpha + O(\beta_n)$.

Remark: Although our definition permits $\{\alpha_n\}_{n=1}^{\infty}$ to be compared with an ARBITRARY sequence $\{\beta_n\}_{n=1}^{\infty}$, in nearly every case we use

$$\beta_n = \frac{1}{n^p},$$

for some number $p > 0$. We are generally interested in the LARGEST value of p with $\alpha_n = \alpha + O(\frac{1}{n^p})$.

Math 128A, Summer 2019
Lecture 4, Thursday 6/27/2019

1 Reviewing Homework 1 Problems

Homework problem 1: $f(x) := \sum_{j=1}^{n} f_j(xy_j)^k$. This problem is very basic in pieces, but it tells us that when we want to do a fourier transform, the fact that we can move the x out to the left means that the variables are separated and x, y live in different worlds. We can do the fast fourier transform on the condition that the x, y can be separated (separation of variables, aka tensor product).

Homework problem 2:

$$s_n := \sum_{k=1}^{n} \frac{1}{k^2} \to \frac{\pi^2}{6} = (((1 + \frac{1}{4}) + \frac{1}{9}) + \frac{1}{16}) + \cdots$$

(L to R) Other way would be right to left (R to L):

$$\left(\left(\frac{1}{n^2} + \frac{1}{(n-1)^2}\right) + \frac{1}{(n-2)^2}\right) + \cdots + 1$$

Neither ways is best, but R to L intuitively should be better beacuse we're adding small terms before larger, so we don't lose as much accuracy.

Method 1 :
Quote a known result.

$$|fl(S_n) - S_n| \leq \frac{\varepsilon}{2} \cdot (|s_2| + |s_3| + \cdots + |s_n|) \tag{1}$$
$$\leq (n-1)\varepsilon \tag{2}$$

This bound may not be optimal, but it does pay attention to order of summation.

We'll reason that the value in the parenthesis is ≤ 2. The crudest thing to say is above in the second line.

Remark: The above is flawed because

$$fl\left(\frac{1}{k^2}\right) = \frac{1}{k^2}(1 + 2\delta_k)$$

But to fix this, we can bridge the LHS and RHS by using the Triangle Inequality.
(1) Sum exact terms in floating point arithmetic into $\hat{s}_n$, to equal $fl\left(\sum_{k=1}^{n} \frac{1}{k^2}\right)$.
(2) Sum the wrong terms exactly:

$$\left|\sum_{k=1}^{n} \frac{1}{k^2}(1 + 2\delta_k) - S_n\right| = \left|\sum_{k=1}^{n} \frac{1}{k^2} \underbrace{2\delta k}_{\varepsilon}\right| \leq 2\varepsilon$$

The floating point of k^2,

$$fl(k^2) = k^2(1+\delta_k)$$

so,

$$\frac{1}{k^2} = \frac{1}{k^2(1+\delta_k)}(1+\delta_k')$$

In the worst case, the numerator and denominator have opposite signs and accumulate error.

$$\begin{aligned}
&\left| fl\left(\sum_{k=1}^{n} \frac{1}{k^2}(1+2\delta_k)\right) \underbrace{-\hat{S}_n + \hat{S}_n}_{\text{bridge}} - Sn \right| \\
&= (n-1)\varepsilon + \frac{\varepsilon}{2}\left(\left|\sum_{k=1}^{2} \frac{1}{k^2}(1+2\delta_k)\right| + \left|\sum_{k=1}^{3} \frac{1}{k^2}(1+2\delta_k)\right| + \cdots + \left|\sum_{k=1}^{n} \frac{1}{k^2}(1+2\delta_k)\right|\right) \\
&\leq 2(n-1)\varepsilon
\end{aligned}$$

Where each of these summation is ≤ 2.
We add as many bridges in to get what we want (things equal to 0). Or multiply something equiv to 1.

Homework 1 Problem 1b-c (b) Find a polynomial $P(x)$ with complex coefficients such that

$$|P(x) - e^{ix}| \leq \epsilon$$

on the interval $|x| \leq 1$.
(c) Design an algorithm for approximating

$$g(x) := \sum_{j=1}^{n} g_j e^{ixy_j}$$

at n points x_i in $O(n)$ operations, with absolute error bounded by

$$\epsilon \sum_{j=1}^{n} |g_j|.$$

KNOWN FACTS:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, x \in \mathbb{R}$$
$$e^{ix} = \sum_{n=0}^{\infty} \frac{(ix)^n}{n!}, x \in \mathbb{R}$$
$$e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!}, z \in \mathbb{C}, z := x + yi$$
$$e^{\alpha+\beta i} = e^{\alpha}(\cos\beta + i\sin\beta)$$

In our problem 1c, recall that we have even and odd parts, where the real parts are even, odd parts are imaginary. To see this, consider $i, i^2 = -1, i^3 = -i, i^4 = 0, i^5 = i, \ldots$. The absolute value (complex number) is bounded to a real number (distance).
So taking the absolute value difference of two complex numbers, we have:

$$\left| e^{ix} - \sum_{n=0}^{p-1} \frac{(ix)^n}{n!} \right| = \left| \sum_{n=p}^{\infty} \frac{(ix)^n}{n!} \right|$$
$$= \sum_{n=part}^{\infty} \left| \frac{(ix)^n}{n!} \right| = \sum_{n=p}^{\infty} \frac{|i|^n x^n}{n!}$$
$$\leq \sum_{n=p}^{\infty} \left(\frac{e}{n}\right)^n |x|^n \leq \sum_{n=p}^{\infty} \left(\frac{e|x|}{n}\right)^n$$

n is going from $p \to \infty$, so we can fix this by replacing n with

$$\leq \sum_{n=p}^{\infty} \left(\frac{e|x|}{p}\right)^n \quad \text{(which is a geometric series)}$$
$$\leq 2 \cdot 2^{-p} \quad (\text{if } |x| \leq 1, p \geq 6)$$

New Example: Like our homework 1.3, consider:

$$a \leq fl\left(\frac{a+b}{2}\right) \leq b, \text{ if } a \leq b$$
$$0 < a \leq fl\left(\sqrt{ab}\right) \leq b$$

The above is the arithmetic mean, whereas the bottom is the geometric mean. The implications of this are applied to within an algorithm, where our numbers are floating point numbers.
Consider:

$$\begin{aligned} a &\leq a \\ a &\leq b \\ 2a &\leq a+b \quad \text{(add ineqs)} \\ fl(2a) &\leq fl(a+b) \quad \text{(what happens when we divide by 2?)} \\ \frac{fl(2a)}{2} &\leq \frac{fl(a+b)}{2} \quad \text{(fine to divide by 2)} \end{aligned}$$

Note rounding is monotone, which helps us know what happens when we perform operations and round (floating point arithmetic).
Consider:

$$\begin{aligned} a^2 &\leq ab \\ \sqrt{a^2} &\leq \sqrt{ab} \\ fl\left(\sqrt{ab}\right) &= \sqrt{ab(1+\delta)} \\ &= \sqrt{ab}(1+\underbrace{\frac{\delta_1}{2}}_{\leq \frac{\varepsilon}{4}}) + O(\varepsilon^2) \end{aligned}$$

Aside:

$$\sqrt{1+\delta} = 1 + \frac{x}{2} - \frac{x^2}{4} + \cdots \quad \text{(Taylor expansion)}$$

Also consider:

$$|\sqrt{1+x}-1| = \left|\frac{(\sqrt{1+x})(\sqrt{1+x}+1)}{\sqrt{1+x}+1}\right| = \left|\frac{x}{1+\sqrt{1+x}}\right| \leq O(|x|)$$

Break time:

2 Bisection

Now, the goal is to solve the following, using bisection (the simplest possible algorithm). Next week we'll talk about variations of Newton and FixedPoint Iteration (which does not guarantee an existing solution), but are higher-dimension.
For Bisection, we apply the intermediate value theorem (IVT). If $f(a) \leq 0 \leq f(b)$, or the other way around $f(b) \leq 0 \leq f(a)$, then this implies

$$\exists_{x \in [a,b]} f(x) = 0,$$

assuming $f \in C[a,b]$; that is, assuming f is continuous over $[a,b] \in \mathbb{R}$.
Our plan:
Step 0: Bracket the root; find a, b with $f(a)f(b) < 0$ (opposite signs).
$f(x) := \ln x$, $\frac{1}{e} < x < e$, $\ln \in (-1,1), x = 1$.
Then we refine the bracket:
Take $m = \text{middle}(a,b)$, with one of the following definitions:

$$\begin{aligned} \text{mid}(a,b) &= \frac{a+b}{2} \quad \text{(tradition but stupid)} \\ &= \sqrt{ab} \quad \text{(better)} \\ &= \max\{a, \min\{b, \sqrt{ab}\}\} \quad \text{(if } a,b > 0\text{)} \end{aligned}$$

2.1 Pseudocode / Algorithm

If $f(m) = 0$, then stop.
If sign $f(m) =$ sign $f(a)$, replace $a := m$, then $f(a) := f(m)$. Otherwise, $b := m$.
Recursion.

Stopping Criterion:
What if we're given a user-specified tolerance `tol`? And let $|b-a| \leq$ `tol`?
How about stopping when $|b-a| \leq \varepsilon \min\{|b|, |a|\}$?
How about stopping when the floating point numbers are equal, $m = a$ or $m = b$? This isn't too bad.
Some people also stop when $|f(m)| < \varepsilon \cdot \min\{|f(a)|, |f(b)|\}$.

Aside: anything like user-specified or provided by the user is never a good idea (it won't sell if it needs the user to perform the hard work).

How to bracket $[a,b]$?
- Randomness if ignorant
- Structural if knowledgeable about f

2.2 Bisection in Real (Infinite) Arithmetic

$$(b-a) \to \frac{1}{2}(b-a) \to \frac{1}{4}(b-a) \to \cdots$$

2.3 Bisection in Floating Point Arithmetic

Stopping criterion of $m = a \cup m = b$. In computer arithmetic, we are computing some $64 - bit$ number. Bisection should produce 1-bit per step, so ≤ 64 steps.
Theoretically, trying to find some 64-bit number, asking 64 Yes/No questions should find us what we want.
In our homework, we are converging down to 0, and there are actually a lot of floating point numbers at 0. Turns out it takes like 1000 steps to underflow to get this stopping criterion (equality), because $2^{-1074} =$ smallest nonzero floating point number.

Example: Let $f(x) = x = 0$, with $[a,b] := [-1, 2]$, and $m := \frac{a+b}{2}$.
So we have:
$m = \frac{-1+2}{2} = \frac{1}{2}$, then:

1: $[a, b] = [-1, \frac{1}{2}]; m = \frac{-1+1/2}{2} = \frac{-1}{4}$
2: $[a, b] = [-\frac{1}{4}, \frac{1}{2}]$
$\vdots$
around $k \approx 1074$: we get $[-2^{-k}, 2^{-k+1}]$, until 2^{-k} underflows.
Aside: our homework problem is designed to convince us that using the Arithmetic mean can take about 20 times as long as using the geometric mean.

2.4 Geometric mean

The good part is: $[a, b] = [2^{-1000}, 1]$ with $f(x) := x - 2^{-900}$.
$m := \sqrt{ab} = \sqrt{2^{-1000}} = 2^{-500}$
Now we're 500 orders of magnitude closer to the solution, and better yet, we have 1 bit of the solution. What if $a = 0$? What if $a < 0 < b$?
So we need a little more detail in the logic. Our first observation is

$$\begin{aligned}
&0 < a \le b \quad \text{then let } m := \sqrt{ab} \\
&a \le b < 0 \quad \text{then let } m := -\sqrt{ab} \\
&0 = a \le b \quad \text{then let } m := \texttt{realmin} = 2^{-1074} \\
&a \le b = 0 \quad \text{then let } m := -\texttt{realmin} = -2^{-1074} \\
&a < 0 < b \quad \text{then let } m := 0
\end{aligned}$$

Remark: In general, we'd rather take a few extra steps via GM than take potentially thousands more steps via AM. (Geometric mean, Arithmetic mean).

Math 128A, Summer 2019
Lecture 5, Monday 7/1/2019

Announcements: Homework due in class Wednesday.

1 Homework 1 Review (due Wednesday in Class)

1.1 Problem 1

$(Fg)_j = \sum_{k=1}^{n} \underbrace{e^{ix_j y_k}}_{F_{ij}} g_k$. We denote F_{jk} for fourier, although this acts like Laplace transform, for things like decay rate. (edit: nevermind, we inserted the i, so it's legal to call this F for fourier transform.)

We need $O(n^2)$ cost to get the matrix elements of F_{jk} because each x_j and y_k has to talk to each other. To see this, recall that matrix multiplication is like dotting two vectors; n multiplications and n additions, per row. So in total for the matrix, we need $O(n^2)$.

The essence of the problem is that we can do this faster if we exploit the fact that we can bring out a term. Like in (a) where we let

$$\sum_{j=1}^{n} (x_i y_j)^k f_j = x_i^k.$$

So we have:

$$\begin{bmatrix} e^{ix_1 y_i} & \dots & e^{ix_1 y_n} \\ \vdots & \ddots & \vdots \\ e^{ix_n y_1} & \dots & e^{ix_n y_n} \end{bmatrix}$$

This problem is about a "miracle" that rank $B \leq r$. Suppose our matrix C is $r \times n$, with $r = 15 << n$; thus $O(15n) = O(rn) = O(n) << O(n^2)$. 15 (or the actual number) is a lot, but it is surely less than n^2.

Hence r depends on our accuracy ε and not on n. So for the matrices $F, B := CD, E =$ error, we have:

$$\mathcal{M}(F) = \underbrace{\mathcal{M}(CD)}_{\mathcal{M}(B)} + \mathcal{M}(E)$$

$$\text{rank } [(n \times r)(r \times n)] = \min\{(n \times r), (r \times n)\}$$

So to ensure the error bound of each element, in this problem we use:

$$\begin{aligned} e^{ix_j y_k} &= \sum_{p=0}^{r-1} \frac{(ix_j y_k)^p}{p!} + \overbrace{O\left(\frac{1}{r!}\right)}^{E} \\ &= \sum_{p=0}^{r-1} \left(\frac{ix_j^p}{\sqrt{p!}}\right)\left[\frac{y_k^p}{\sqrt{p!}}\right] \quad \text{(or equivalently can split by } 1 \cdot p!) \\ &= \sum_{p=0}^{r-1} (C_{jp}) \cdot [D_{pk}] \end{aligned}$$

The miracle is that we usually, in a fourier transform, have:

$$U(x,t) = \sum_{j=0}^{\infty} f_j(t) g_j(x)$$

where we need infinite sums because one of f_j or g_j is not enough.

1.2 Problem 2

(in class) The maximum accuracy achievable (in computing $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$):

$$fl\left(\sum_{k=1}^{n} \frac{1}{k^2}\right) = s_n\left(1 + 2n\delta\right), \qquad |\delta| \leq \varepsilon$$

There are two competing errors: truncation error on the left, and the floating point error in summing to infinity. Our question is to find the minimum error in between these two.
So the maximum accuracy at the bottom of this curve will be $O(n\varepsilon) + O\left(\frac{1}{n}\right)$, where $\sqrt{1/\varepsilon} = 2^{26}$. The best thing would be to take 2^{26}, and our value of accuracy will only be 2^{-26}.

2 Fixed Point Iteration

- Bisection (bicycle; low cost, doesn't smell)
- Fixed point iteration (car)
- Newton's Method (maseratti; good, but requires maintenance or breaks down)

Step 0: Convert to fixed point form. That is, convert $f(x) = 0 \to x = g(x)$. For example,

$$\begin{aligned} f(x) = 0 &\to x = x - f(x) = g(x) \\ &\to x = x + \alpha f(x) \\ &\to x = x + h(x) f(x) \end{aligned}$$

For choosing, we question if we want to introduce extraneous solutions and how fast we reach convergence

2.1 Example of Fixed Point Iteration in Action (and Error Bounding via MVT)

$$\underbrace{xe^x - 1 = 0}_{f(x)}, x = w(1) \quad \text{(Lambert's something)}$$

$$xe^x = 1$$
$$x = e^{-x} = g_A(x)$$

Alternatively, we could use $e^x = \frac{1}{x}$ or $x = \ln\left(\frac{1}{x}\right)$.
Step 1: Guess some x_0
Step 2: Let $x_1 := g(x_0)$; that is, project $g(x_0)$ to $y = x$ line and set it as the new input.
Step 3: Keep going with $x_{k+1} := g(x_k)$.
Or shown differently:

$$x_0 \mapsto x_1 = g(x_0) \mapsto x_2 = g(x_1) \mapsto x_3 = g(x_2) \mapsto \cdots \mapsto x_{n+1} = g(x_n)$$

Remark: These seem to converge (spiral inwards), but how do we conduct (numerical) analysis?

(0) Correct limit?
For example, $x_{n+1} = g(x_n)$. So suppose $x_n \to x$. Then $g(x_n) \to g(x)$ because g is continuous. Also, $x_{n+1} \to x$. Then $x \leftarrow x_{n+1} = g(x_n) \to g(x)$.

So if $x_n \to x$, as $n \to \infty$, then $x = g(x)$ is a solution. So if $x = e^x$, then we reverse the steps:

$$xe^x = 1 \implies xe^x - 1 = f(x) = 0$$

is a solution. We cannot blanketly say this is the unique solution, because our original equation may have multiple (or extraneous) solutions. But we can be sure that our algorithm (fixed point iteration) does converge to the correct thing, provided the limit exists.

(1) Rate of convergence?
Assuming (or given) g differentiable:

$$x_{n+1} = g(x_n)$$
$$x = g(x)$$
$$\underbrace{x_{n+1} - x}_{e_{n+1}} = g(x_n) - g(x) \quad \text{(subtracting these equations, we get the error)}$$
$$\neq g(e_n)$$
$$= g'\underbrace{(\xi_n)}(x_n - x)$$

where this ξ_n is unknown, given by MVT between x_n and x

$$\implies |e_{n+1}| \leq |g'(\xi_n)||e_n|$$

This is an optimistic bound; usually, the error is not (much) lower than this upper bound. We use this to estimate:

$$|e_{n+1}| \leq |g'(\xi_n)||e_n| \leq \max|g'(x)||e_n|$$

Consider:

$$\begin{aligned} g(x) &= e^{-x} \\ g'(x) &= -e^{-x} \\ |g'(x)| &= |-e^{-x}| = e^{-x} \leq \frac{1}{2}; \quad x \geq \ln 2 \approx 0.6931 \end{aligned}$$

However, $x = e^{-x} \leq \frac{1}{2}$. We call this "slightly slow convergence". (To estimate this precisely, we need to know the solution; but of course, we are trying to *find* the solution, so this is non-optimal.)

So we try something else:

$$\begin{aligned} xe^x - 1 &= 0 \\ x + xe^x - 1 &= x \\ x(1 + e^x) &= 1 + x \\ x &= \frac{1+x}{1+e^x} =: g(x) \end{aligned}$$

This should be equivalent because we didn't divide by 0. So for the derivative,

$$\begin{aligned} g'(x) &= \frac{(1+e^x)\cdot 1 - (1+x)\cdot e^x}{(1+e^x)^2} \\ &= \frac{e^x + 1 - e^x - xe^x}{(1+e^x)^2} \\ &= \frac{1 - xe^x}{(1+e^x)^2} \end{aligned}$$

So $g'(x) = 0$ at the solution; so if we get close enough within $\pm\frac{1}{2}$ of the solution, it is guaranteed that we will find the solution.

What happens if we try to bound:

$$\left|\frac{1-xe^x}{(1+e^x)^2}\right| \leq \begin{cases} \frac{1}{4} & x = 0 \\ \text{maybe } \frac{1}{2}? & 0 < x < 1 \\ \frac{1}{5} & x = 1 \end{cases}$$

To prove this, suppose $0 < x_0 < 1$. Then

$$0 \leq g(x_0) = x_1 = \frac{1+x_0}{1+e^{x_0}} \leq 1$$

which we verify by comparing the graphs of $e^x > x$ in this interval, and knowing that e^x is positive.

Now consider:

$$|g'(x)| = \underbrace{\left|\frac{1-xe^x}{(1+e^x)^2}\right|}_{\text{find upper bound}} \leq \frac{1+xe^x}{(1+e^x)^2} \leq \frac{1+e}{4} \leq \frac{3.7}{4} < 1$$

This isn't good enough, so consider that the minimum of the denominator is 2^2 (at $x = 0$).

For the numerator, we check endpoints and where the derivative of the numerator is 0:

$$|1 - xe^x| \leq \begin{cases} 1 & x = 0 \\ e - 1 \approx 1.7 & x = 1 \end{cases}$$
$$\implies -1 \cdot e^x - xe^x < 0$$

So this shows that $|1 - xe^x| \leq 1.7$, so we showed our desired expression is bounded by $\frac{1.7}{4} < \frac{1}{2}$.

Conclusion. This $g(x)$ is better than $g(x) = e^{-x}$.

Theorem 2.1. Suppose $x_{n+1} = g(x_n)$, with g continuous.
1. $a \leq x \leq b \implies a \leq g(x) \leq b$; Invariance
2. $a \leq x \leq b \implies |g'(x)| \leq |\frac{1}{2}|$; Contraction

then $|x_n - x| \leq 2^{-n}|x_0 - x|$ and $x_n \to x$ as $n \to \infty$

Famous example.
The square root $\sqrt{}$.

Step 0:

$x_{n+1} := \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) = g(x_n)$. If $x_n \to x$, then $x = g(x)$.

$$x = \frac{1}{2}\left(x + \frac{a}{x}\right)$$
$$2x = x + \frac{a}{x}$$
$$x = \frac{a}{x}$$
$$x^2 = a \implies x^2 - a = 0.$$

Remark: Usually easier to check contraction first, then invariance after.

Step 1: Contraction

$$|g'(x)| = \left|\frac{1}{2}\left(1 - \frac{a}{x^2}\right)\right| \leq \frac{1}{2}$$
$$\implies \left|1 - \frac{a}{x^2}\right| \leq 1$$
$$\implies -1 \leq 1 - \frac{a}{x^2} \leq 1.$$

UB is obvious, so we check LB:

$$-1 \leq 1 - \frac{a}{x^2}$$
$$-2 \leq -\frac{a}{x^2}$$
$$2 \geq \frac{a}{x^2}$$
$$x^2 \geq \frac{a}{2}$$
$$x \geq \sqrt{\frac{a}{2}} = \frac{\sqrt{a}}{\sqrt{2}}$$

We can then assume $1 \leq a \leq 4$; as long as we start as bigger than $\sqrt{a}$, for example a, then we are good. So we have shown contraction.
Step 2: Invariance

$$g(x) = \frac{1}{2}\left(x + \frac{a}{x}\right)$$

To prove this is invariant, let's try the interval $1 \leq x \implies g(x) \geq \frac{1}{2}\left(1 + \frac{a}{a}\right) = 1$. Then $x \leq a \implies g(x) \leq \frac{1}{2}\left(a + \frac{a}{1}\right) = a$, so the interval $[1, a]$ is invariant (quite beautifully).
Alternatively, we could use calculus with $g'(x) = \frac{1}{2}\left(1 - \frac{a}{x^2}\right) = 0$ at $x = \sqrt{a}$.

Remark: So $[1, a]$ is invariant and $|g'(x)| \leq \frac{1}{2}$ over this interval, so $|x_n - x = \sqrt{a}| \leq 2^{-n}|a - 1|$.

Lecture ends here. Next time we'll talk more about Newton's method and fixed point iteration.

Math 128A, Summer 2019
Lecture 6, Tuesday 7/2/2019

Today's Agenda:
- Parameters
- fl (Fixed point iteration)
- Quadratic Convergence

1 On Parameters and Function Handles

```
1
2 [r,h] = bisection(a,b, f)
3
4   y = bisection(a,b, @f, p)
5
6 function y = f(x,p)
7   y = (x - eps ** 3) ** 3
8   or   if  p == 0
9            y = 0
10       if  p == 1
11           y = 1
12       if  p > 1
13           y = f(x, p-1) + f(x, p - 2)
14  end
```

We call `@f` a "function handle". We can use parameters in the above way as `p`. We can pass through optional parameters.
For Homework 1, Problem 6, we need to
- (Write and) Print a program
- Print Results (save into a text file; matlab: `diary`, screenshot, or picture)
- Include Plots

2 Floating Point: Fixed Point Iteration

[1] Solve $f(x) = 0 \iff x = g(x)$. One obvious way is to add x to both sides of the equation, but it turns out this is usually a very bad idea.
[2] Come up with better ways to create $g(x)$.
[3] Then, iterate $x_{n+1} = g(x_n)$ until satisfied (the exact number of digits; within tolerance) of the exact solution $x = g(x)$. That is, we stop when

$$\frac{|x_n - g(x_n)|}{|x_n|} \leq \varepsilon \qquad \text{if } x_n \neq 0$$

Aside: In some institutions, our tolerance (RHS) is user-specified. But for us, we say anything user-specified is bad, so we use the absolute ε.

Yesterday, we proved the following theorem:

Theorem 2.1. If $a \leq x \leq b$ implies:
1. $a \leq g(x) \leq b$ [Invariant]
2. $|g'(x)| \leq \frac{1}{2}$ [Contractive],

then for any $x_n \in [a, b]$, we have:

$$x_{n+1} = g(x_n); x = g(x) \implies |x_n - x| \leq 2^{-x}|x_0 - x|$$

Moreover, this gives a **unique** solution.

For some insight, consider:

$$\begin{aligned} x_{n+1} - x = |e_{n+1}| &= |g'(\xi_n)e_n| \\ &\leq \frac{1}{2}|e_n| \\ &\vdots \\ &2^{-(n+1)}|e_0| \end{aligned}$$

Example: $\sqrt{\text{something}}$: It turns out that in our example $\sqrt{}$, we can get quadratic convergence, because as we get closer to our solution, we do so at an increasing rate.

$$\begin{aligned} \sqrt{}: \qquad g(x) &= \frac{1}{2}\left(x + \frac{a}{x}\right), \qquad \text{if } 1 \leq a \leq 2 \\ g'(x) &= \frac{1}{2}\left(1 - \frac{a}{x^2}\right) = 0, \qquad \text{if } x = \sqrt{a}. \end{aligned}$$

2.1 Showing Uniqueness of Solution

Suppose $x = g(x)$ and $y = g(y)$. Note that we aren't supposing anything for contradiction; to show uniqueness, we simply show these have to be equal. Consider:

$$\begin{aligned} |x - y| &= |g(x) - g(y)| \\ &= |g'(\xi)(x - y)| \\ &\leq \frac{1}{2}|x - y| \end{aligned}$$

Additionally, let $z := |x - y|$.

$$0 \leq z \leq \frac{1}{2}z \implies \frac{1}{2}z \leq 0 \implies 0 \leq z \leq 0$$

Hence $z = 0$, and $|x - y| = 0 \implies x = y$.

2.2 What happens in Floating Point Arithmetic?

Let's say we're trying to solve $0 = f(x) \iff x = g(x)$. If we're lucky (if we program it well), in floating point, this will be $g(x)(1 + \delta)$. This is an idealistic forward error. In other words, we deliver a g that is fairly accurate. Otherwise, we can deliver some $g\left(x(1 + \delta)\right)$, which is an **idealistic backward error**.

That is, we have:

$$fl[g(x)] = \begin{cases} g(x)(1+\delta) & \text{idealistic forward error} \\ g[x(1+\delta)] & \text{idealistic backwards error} \end{cases}$$

For example, we have $g(x)(1+17\delta_1)+18\delta_2$ realistically. So we see that:

Fixed point iteration is (somewhat) self error-correcting.

Consider a set of operations stacked within g, and the algorithms are fixed. So we assume that it does the best possible. $y = fl(x_n)$.

$$\begin{aligned} y_{n+1} &= g[y_n](1+\delta_n) \\ x &= g(x) \\ e_{n+1} &= y_{n+1} - x = g[y_n](1+\delta_n) - g(x)(1+\delta_n) + g(x)\delta_n \\ &= (1+\delta_n)\left[g(y_n) - g(x)\right] + \underbrace{x}_{=g(x)}\delta_n \end{aligned}$$

Notice the bridges (tricks with g) we use in the above manipulations to get our result. Now we use the MVT, put absolute values on everything and proceed as usual.

$$|e_{n+1}| \le \frac{1}{2}(1+\varepsilon)|e_n| + |x|\varepsilon$$

Notice the differences from our previous error expression. We should be looking at

$$r_n := \frac{|e_n|}{|x|}, \qquad \text{relative error in } x_n,$$

where

$$r_{n+1} \le \frac{1+\varepsilon}{2} r_n + \varepsilon$$

This is very common where our error at one step is the error at the previous step times some constant, plus an added error (constant). We accordingly call this linear.
Let's call $\theta := \frac{1+\varepsilon}{2}$ (because it looks like a 0 and 1 interval, taking a nap, according to Strain).
Consider from the above, we have the following pattern:

$$\begin{aligned} r_{n+1} &\le \theta r_n + \varepsilon \\ r_n &\le \theta r_{n-1} + \varepsilon \\ r_{n+1} &\le \theta(\theta r_{n-1} + \varepsilon) + \varepsilon \\ &\vdots \\ &\le \theta^{n+1} r_0 + \underbrace{\left(\theta^n + \theta^{n-1} + \cdots + \theta + 1\right)}_{\le 2}\varepsilon \end{aligned}$$

where

$$r_0 = \frac{|x_0 - x|}{|x|} \le \frac{|b-a|}{\min\{|a|,|b|\}}.$$

We verify, $fl\left(\frac{1}{2}+\frac{\varepsilon}{2}\right)=\frac{1}{2}+\frac{\varepsilon}{2}$.
We know this bound ≤ 2 because:

$$1+\theta+\cdots+\theta^n=\frac{1-\theta^{n+1}}{1-\theta}\leq\frac{1}{1-\theta}$$

and

$$r_{n+1}\leq\underbrace{\theta^{n+1}r_0}_{=0}+\underbrace{\frac{1}{1-\theta}}\varepsilon\leq 2\varepsilon$$

We call the right term the "black hole" of error. In total, this is a very good result.
Recall that back when we were summing $\sum\frac{1}{k^2}$ from left $\to$ right, we had $O(n\varepsilon)$, and at best, $O(\sqrt{\varepsilon})$. So if we're using this summation to compute π, then we won't get good results. However, if we define π as the root of a fixed point iteration, then it's a whole different story (we can get good accuracy).
Recall

$$\varepsilon=2^{-52}\implies\sqrt{\varepsilon}=2^{-26}=0.000\cdots00100\cdots0,$$

as opposed to

$$2\epsilon=0.000\cdots0010.$$

3 Quadratic Convergence

We look at $\sqrt{x}$ as an example.

$$\begin{aligned}
g(x)&=\frac{1}{2}\left(x+\frac{a}{x}\right)\\
x_{n+1}&=g(x_n)\\
x&=g(x)\\
\implies x_{k+1}-x&=g(x_n)-g(x)\\
&=g'(\xi_n)(x_n-x)\quad\text{(MVT)}
\end{aligned}$$

With $\xi_n\in(x_n,x)\to x$, if $g'(x)=0$. To capture the fact that our error gets reduced, we use an additional term of the taylor expansion (linear MVT does not suffice).
So we look at the Taylor Remainder:

$$\begin{aligned}
g(x_n)-g(x)&=g(x)+g'(x)(x_n-x)+\frac{1}{2}g''(\xi_n)(x_n-x)^2-g(x)\\
&=g'(x)(x-x_n)+\frac{1}{2}g''(\xi_n)(x-x_n)^2\\
&=\frac{1}{2}g''(\xi_n)(x-x_n)^2\quad\text{(We note that } g'(x)=0 \text{ when } x=g(x).)
\end{aligned}$$

So we have,

$$\begin{aligned}
e_{n+1}&=\frac{1}{2}g''(\xi_n)e_n^2,\qquad\text{or equivalently,}\\
&=\frac{1}{2}\left(g''(\xi_n)g_n\right)e_n
\end{aligned}$$

Example $\sqrt{4}$:

$$\begin{aligned}
x_0 &= 4 \\
x_1 &= \frac{1}{2}\left(4 + \frac{4}{4}\right) = 2.5 \quad \text{(error 0.5)} \\
x_2 &= \frac{1}{2}\left(2.6 + \frac{4}{2.5}\right) = 2.05 \quad \text{(error } 0.05 = 5 \times 10^{-2}\text{)} \\
x_3 &= \frac{1}{2}\left(2.05 + \frac{4}{2.05}\right) = 2.0061 \quad \text{(error } 6 \times 10^{-4}\text{)} \\
x_4 &= 2.0000000093 \quad \text{(error } 9 \times 10^{-8}\text{)} \\
x_5 &= 2.0 \quad \text{(error } 10 \times 10^{-}16\text{)}
\end{aligned}$$

Consider

$$e_{n+1} := \frac{1}{2} g''(\xi_n) e_n^2.$$

If $a \le x \le b$, then $a \le g(x) \le b$ (Invariance),
and $|g''(x)| \le C, \qquad C|x_0 - x| \le 1$ (Contractive).

$$\begin{aligned}
& |e_n| \le \frac{1}{2} C |e_{n-1}|^2 \\
\implies\ & e_{n+1} \le \frac{1}{2} C |e_n|^2 \le \frac{1}{2} C \left(\frac{1}{2} C |e_{n-1}|^2\right)^2 \\
&= \left(\frac{1}{2} C\right)^3 \cdot |e_{n-1}|^4 \\
&\le \left(\frac{1}{2} C\right)^3 \left(\frac{1}{2} C |e_n - 2|^2\right)^4 \\
&= \left(\frac{1}{2} C\right)^7 |e_{n-1}|^8
\end{aligned}$$

Then,

$$\begin{aligned}
|e_n| &\le \left(\frac{1}{2} C\right)^{2-1} |e_0|^{2^n} \\
&= \frac{2}{C} \left(\frac{C|e_0|}{2}\right)^{2^n}
\end{aligned}$$

This is either great or terrible. It's wonderful if the expression in the parenthesis is close to $\frac{1}{2}$, and terrible for when that is close to 1 (will explode). So we say:

$$\begin{cases} \text{fast convergence} & \text{if } C|e_0| \le 1 \\ \text{divergence} & \text{if } C|e_0| \ge 2 \end{cases}$$

$$\begin{aligned}
g'(x) &= \frac{1}{2}\left(1 - \frac{a}{x^2}\right) \\
g''(x) &= \frac{1}{2}\frac{2a}{x^3} = \frac{a}{x^3}
\end{aligned}$$

So this gives

$$\left|\frac{a|x_0 - x|}{\min x^3}\right| \leq 1 \to \surd \text{ iteration converges very fast}$$

From our theorem, we don't have anything for uniqueness, so we assume $|g'(x)| \leq \frac{1}{2}$ for $a \leq x \leq b$, and $x = g(x) \implies g'(x) = 0$. We call this quadratic convergence because the error is small and being squared (quad for square).

Theorem 3.1. If $(a \leq x \leq b)$ implies:

1. $a \leq g(x) \leq b$,
2. $|g'(x)| \leq \frac{1}{2}$,
3. $|g''(x)| \leq C$, where $C|b - a| \leq 1$
4. $x = g(x) \implies g'(x) = 0 \implies x_n \to x$ and

$$|x_{n+1} - x| \leq \frac{1}{2}C|x_n - x|^2$$

3.1 How do we make $g'(x) = 0$ at $x = g(x)$?

Try $f(x) = 0$. Recall that g is what we design in trying to find roots to $f(x) = 0$. So there's some wiggle-room in how we define or design $g(x)$.

$$x = x + h(x)f(x) =: g(x)$$

And choose $h(x)$ such that $g'(x) = 0$ at the solution x. Differentiating, we get

$$g'(x) = 1 + h'(x) \underbrace{f(x)}_{0 \text{ at } x=0} + h(x)f'(x).$$

We want this $g'(x)$ to be 0 when $f(x) = 0$. So we just need $1 = hf'$, which says we should choose:

$$h(x) := \frac{-1}{f'(x)}.$$

So from

$$g(x) = x - \frac{f(x)}{f'(x)},$$

we have

$$\begin{aligned} g'(x) &= 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} \\ &= 1 - 1 + f(x)\frac{f''(x)}{f'(x)^2} \\ &= f(x)\frac{f''(x)}{[f'(x)]^2} = 0, \end{aligned}$$

if $f(x) = C$.

Remark: Be careful in trying to find roots of something like $f(x) = x^2$, where a slight perturbation of our data (slightly flawed or incorrect data) can make the zero cease to exist (if the parabola shifts up a very slight amount). Instead, we should be looking for a minimum of the function.

If $g(x) - x - \frac{f(x)}{f'(x)}$,

$$\begin{aligned} |g'(x)| &= \left| 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} \right| \\ &= \left| f(x)\frac{f''(x)}{[f'(x)]^2} \right| \leq \frac{1}{2}, \qquad \text{close enough to } x \text{ .} \\ |g''(x)| &= f'\frac{f''(x)}{[f'(x)]^2} + ff'''\frac{1}{[f']^2} + ff''\frac{-2f''}{[f']^3} \end{aligned}$$

So if $\left|\frac{f''(x)}{f'(x)}\right| \leq k$, then we are probably ok.

3.2 Alternate Derivation of Newton's Method

We can expand and operate.

$$\begin{aligned} f(x) &= 0 \\ f(x_n) &\neq 0 \end{aligned}$$

But maybe we can expand $f(x)$ about x_n:

$$\begin{aligned} f(x) &= f(x_n) + f'(x_n)\underbrace{(x - x_n)}_{\text{solve for}} + \frac{1}{2}f''(\xi_n)(x - x_n)^2 \\ \implies x - x_n &= \frac{-f(x_n)}{f'(x_n)}\left(-\frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}(x - x_n)^2\right) \\ x &= x_n - \frac{f(x_n)}{f'(x_n)}\left(-\frac{1}{2}\frac{f''(\xi_n)}{f'(x_n)}(x - x_n)^2\right) \end{aligned}$$

This gives us Newton's method, but we didn't need to know anything from Fixed Point iteration.

Lecture ends here.

Math 128A, Summer 2019
Lecture 7, Wednesday 7/3/2019

CLASS ANNOUNCEMENTS: As tomorrow, Thursday, is July 4th Holiday, no lecture.
Plans for today:

- Newton's Method
- Convergence Theorem
- Multiple Roots
  - Double Stepping
  - Line Search
  - Schroder

1 Newton's Method

View Newton's Method as the inverse of first order approximation (via Taylor). That is,

$$f(x) = 0 = f(x_n) + f'(x_n)(x - x_n) + \left[\frac{1}{2}f''(\xi_n)(x - x_n)^2\right],$$

where we drop the bracketed term, and $\min\{x, x_n\} \leq \xi_n \leq \max\{x, x_n\}$. Solving for x, we have:

$$\begin{aligned} -f(x_n) &= f'(x_n)(x_n - x) \\ -\frac{f(x_n)}{f'(x_n)} &= x_{n+1} - x_n \end{aligned}$$

We usually prefer to write Newton's method as above, as opposed to how we write fixed point iteration:

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

Remark: For error analysis (in numerical analysis in general), we always try to subtract similar equations:

$$\begin{aligned} & 0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \\ - \quad & 0 = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2}f''(\xi_n)(x - x_n)^2 \end{aligned}$$

Subtracting these give:

$$0 = f'(x_n)(x_n)\underbrace{(x_{n+1} - x)}_{e_{n+1}} - \frac{1}{2}f''(\xi_n)\underbrace{(x - x_n)^2}_{e_n^2}$$

Where on the left we have the new error, and on the right we have the old error, squared.

Thus we have:

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

As mentioned before, either this gets accurate very quickly or goes bad quickly (which would feed us NaN early, rather than later).
Dividing by the solution, then we have the **relative error**.

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

Recall that relative is dimensionless, so we can express these as percentages (whereas for absolute error, we have to specify units like inch). Everyone knows what we're talking about when we say 0.7% error.

Definition: Dimension -

We define brackets around something to denote its dimension, as such.

$$\left[\frac{d^2f}{dx^2}\right] =: [f''] := \frac{F}{x^2}$$

$$\frac{xF/x^2}{F/x} = 1$$

For example, let $f(x)$ have units therbligs (lightyears).
F x
We have $f(x) \iff 10^6 f(10^{-6}x) = 0$.
If $f : \mathbb{R}^n \to \mathbb{R}^n$, then $Df(x)$ is the $n \times n$ matrix,

$$\left[\frac{\partial f_i}{\partial x_j}\right].$$

So if $f(x,y) = \begin{pmatrix} f_1(x,y) & f_2(x,y) \\ x^2 - y^2, & 2xy \end{pmatrix}$, then

$$[Df] = \begin{bmatrix} 2x & -2y \\ 2y & 2x \end{bmatrix}$$

1.1 2 Dimensional Newton's Method (Aside)

We won't use this in this course, but just so we know it's the same as one dimention:

$$f(x) \cong f(x_n) + Df(x_n)(x_n - x) = 0$$
$$-f(x_n) = Df(x_n)(x_{n+1} - x_n)$$

On the left and on the right, we have column vectors, where $Df(x_n)$ is a square matrix.
Then we have:

$$x_{n+1} = x_n - Df(x_n)^{-1}f(x_n)$$

Usually we can't invert the jacobian because it depends on x, but we can get close to it by using \ (backslash) in Matlab.

1.2 Back to 1 Dimension

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

Theorem 1.1. Convergence Theorem for Newton's Method: Suppose $f(x) = 0$, and $\frac{|x-x_0|}{|x|} \leq c$ (the relative error is invariant), where

$$\max_s f''(s) \cdot \max_s \frac{s}{f'(s)} \leq \frac{1}{c}$$
$$\max_{|x-s|\leq c|x|} [f''(s)] \cdot \max_{|x-s|\leq c|x|} \left[\frac{s}{f'(s)}\right] \leq \frac{1}{c}.$$

If these are satisfied, then Newton's method x_n converges quadratically to x.

1.3 Example

Recall our example of the square root.

$$\begin{aligned} f(x) &= x^2 - a \\ f'(x) &= 2x \\ x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \end{aligned}$$

Using the theorem above,

$$x^2 - a = 0, \qquad \frac{|x - x_0|}{|x|} \leq C = 1$$
$$\max_{|x_s|\leq C|x|} 2 \cdot \max_{|x_s|\leq C|x|} \frac{s}{2s} \leq \frac{1}{c} = 1$$

and thus we see we have quadratic convergence from any x_0 with:

$$\begin{aligned} |x - x_0| &\leq |x| \\ |\sqrt{a} - x_0| &\leq \sqrt{a} \\ -\sqrt{a} \leq \sqrt{a} - x_0 &\leq \sqrt{a} \\ -2\sqrt{a} &\leq -x_0 \\ 2\sqrt{a} &\geq x_0 \\ \implies x_0 &\leq 2\sqrt{a} \end{aligned}$$

So if $1 \leq a \leq 2$, then $1 \leq \sqrt{a} \leq \sqrt{2}$, which implies ($x_0 \leq 2 \implies$ quadratic convergence).

1.4 Newton's Method for Multiplicity > 1

In this case with multiple case, we have problems because:

$$\begin{aligned} f(x) = 0 = f'(x) = f''(x) = \cdots = 0, \quad f^{(m)}(x) \neq 0 \\ f(x) = x^3 \\ f'(x) = 3x^2 \\ f''(x) = 6x \\ f^{(3)} = 6 \end{aligned}$$

This causes issues because Newton's method says:

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

We can be more specific, via Taylor:
Let $g(y) := \frac{f^{(m)}(x)}{m!} + \frac{f^{(m+1)}(x)(y-x)}{(m+1)!} + \cdots$.

$$\begin{aligned} f(y) &= f(x) + f'(x)(y-x) + \cdots + \frac{f^{(m-1)}(x)(y-x)^{m-1}}{(m-1)!} + \frac{f^{(m)}(x)(y-x)^m}{m!} + \cdots \\ &= g'(y)(y-x)^m + g(y)m(y-x)^{m-1} \\ &= \frac{g(y)\cdot(y-x)^m}{g'(y)\cdot(y-x)^m + m\cdot g(y)\cdot(y-x)^{m-1}} \\ &= \frac{g(y)\cdot(y-x)}{g'(y)\cdot(y-x) + m\cdot g(y)} \to \frac{0}{m\cdot g(y)} \end{aligned}$$

Then,

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{g(x_n)\cdot\overbrace{(x_n - x)}^{e_n}}{g'(x_n)\cdot\underbrace{(x_n-x)}_{e_n} + m\cdot g(x_n)} \\ x &= x \\ e_{n+1} &= e_n - \frac{g(x_n)e_n}{g'(x_n)\cdot e_n + m\cdot g(x_n)} \\ &= e_n - \frac{1}{m}e_n + O\left(e_n^2\right) \end{aligned}$$

To see the first line for e_{n+1}, consider $m\cdot g(x_n)$ is nonzero, so

$$\begin{aligned} \frac{g(x_n)}{m\cdot g(x_n) + g'(x_n)\cdot e_n} &= \frac{g(x_n)}{g(x_n)\cdot g\left(1 + \frac{g'(x_n)}{m\cdot g(x_n)}e_n\right)} \\ &= \frac{1}{m}\left(1 - \frac{g'(x_n)}{m\cdot g(x_n)}e_n + O\left(e_n\right)^2\right) \end{aligned}$$

Where we have a geometric series, hence the last line.
WHen $m = 3$, then

$$|e_{n+1}| \leq \frac{2}{3}|e_n| + O\left(e_n^2\right)$$

So Newton fails to have quadratic convergence, if $m \geq 2$. Using the secant method never gets (as good as) quadratic convergence, but it can be safe. Or, we can try to fix Newton's method somehow, using **double stepping**. That is,

$$x_{n+1} = x_n - \underbrace{m}_{\neq 1} \frac{f(x_n)}{f'(x_n)}$$

This would cancel the $\frac{1}{m}$ from above, where m is the multiplicity of the root. The downside, is practically we don't know m (but we need m), but we know that Newton's method is just not working. The upside, is we can find m by:

1.5 Option 1:

- Take 1 step, with $m = 1$, which gives $x_{n+1}^1 = x_n - \underbrace{\frac{f(x_n)}{f'(x_n)}}_{=: \Delta_n}$
- Try $m = 2$, which gives $x_{n+1}^2 = x_{n+1}^1 - \Delta_n$.
- $\vdots$
- $x_{n+1}^k = x_{n+1}^{k-1} - \Delta_n$.

Choose $i \leq j \leq k$, so

$$|f(x_{n+1}^2)| = \min_{1 \leq j \leq k} |f(x_{n+1}^j)|$$

This determines m automatically, which costs a few more f evaluation, but k is rarely larger than 3.

1.6 Option 2:

This is to generalize this a bit more, which is that we are checking intergral values:

$$x_{n+1} = x_n - t\frac{f(x_n)}{f'(x_n)},$$

We choose t with $0 \leq t < \infty$ to minimize $|f(x_{n+1}^t)| \leq \frac{1}{2}|f(x_n)|$. We take the Newton direction, then we find a better approach.

1.7 Option 3:

Estimate m from $f(x_n), f'(x_n), f''(x_n)$. So,

$$\begin{aligned}
f(y) &= (y-x)^m \cdot g(y) \\
f'(y) &= (y-x)^m \cdot g'(y) + m(y-x)^{m-1} g(y) \\
f''(y) &= (y-x)^m \cdot g''(y) + 2m(y-x)^{m-1} g'(y) + m(m-1)(y-x)^{m-2} \cdot g(y)
\end{aligned}$$

We want to choose the largest (right-most) terms of the sequence (as we let $y \to x$). So consider:

$$\begin{aligned}
f \cdot f''(y) &= m(m-1)(y-x)^{2m-2} g(y)^2 + O(y-x)^{2m-1} \\
(f')^2 &= m^2 (y-x)^{2m-2} g(y)^2 + \cdots
\end{aligned}$$

Subtracting these two equations, we have:

$$(f')^2 = m(y-x)^{2m-2} g(y)^2 + \cdots$$

Hence, as $y \to x$,

$$\frac{(f')^2}{(f')^2 - f \cdot f''} \to m$$

2 Newton-Schroder

$$x_{n+1} = x_n - \frac{[f'(x_n)]^2}{[f'(x_n)]^2 - f(x_n)\underbrace{f''(x_n)}} \cdot \frac{f(x_n)}{f'(x_n)}$$

The only cost here is to compute $f''(x_n)$. Of course, it's harder to explain what this means in multi-dimensional settings.
This is quadratically convergent, for simple roots **and** roots with multiplicity. So this is a pretty fail-safe way of conducting Newton's method.

$$\begin{aligned}
\left(\frac{f(x)}{f'(x)}\right)' &= 1 - \frac{f(x)f''(x)}{[f'(x)]^2} \\
&= \frac{f'(x)^2 - f(x) \cdot f''(x)}{[f'(x)]^2}
\end{aligned}$$

This would suggest that Newton-Schroder's method is

$$\begin{aligned}
x_{n+1} &= x_n - \left[\frac{f(x_n)}{f'(x_n)}\right] / \left[\frac{f(x_n)}{f'(x_n)}\right]' \\
&= x_n - \frac{F(x_n)}{F'(x_n)},
\end{aligned}$$

where we let

$$F(y) = \frac{f(y)}{f'(y)} = \frac{(y-x)^m g(y)}{(y-x)^m \cdot g'(y) + m \cdot (y-x)^{m-1} \cdot g(y)}$$

Remark: This is interesting because F no longer has a zero with multiplicity more than one, at $y = x$. The numerator $f(y)$ has a zero of multiplicity m, but the denominator has a zero with multiplicity $(m-1)$.

Math 128A, Summer 2019

Lecture 8 Thursday, 7/4/2019

CLASS ANNOUNCEMENTS: July 4 - Independence Day Holiday - NO LECTURE

Remark: This file is a placeholder to preserve the (mod 4) nature of lecture numberings. Refer to next week's lecture 9 on Monday.

Math 128A, Summer 2019
Lecture 9, Monday 7/8/2019

1 Discussion: Homework 2

1.1 Question 4

For 4(b), recall the homework says our interval (α, β) will depend on a.

Aside: Newton's method is affinely invariant (that is, scalar multiplication to $f(x)$ cancels out with its derivative). To see this further, see P. Deuflhard for an in-depth discussion on Newton's method.

For 4(c), consider:

$$g(x) = x(2 - ax)$$
$$|g'(x)| = |2 - 2ax| \leq \frac{1}{2}$$

We're interested in the above line for convergence. So we have:

$$-\frac{1}{2} \leq 2 - 2ax \leq \frac{1}{2}$$
$$\frac{5}{4a} \geq x \geq \frac{3}{4a}$$

So the rate of convergence is quadratic, **because**

$$g'\left(\frac{1}{a}\right) = 0.$$

Aside: In general math, we just care about radius of convergence; however, for applied math, we need fast convergence. Hence we look for a better bound:

$$|g'(x)| \leq \frac{1}{2}$$

1.2 Review: Rate of Convergence

Consider $x_n \to x$. Then we can write

$$|x_n - x| \leq K\theta^n = O(\theta^n), \qquad \theta < 1$$

This gives linear convergence with ratio θ. Sometimes linear convergence is good (and we'd take it where it's **guaranteed**). For example, in bisection, if we start with an interval with the root bracketed.
If our initial error $|x_0 - x| < 1$, then we have quadratic convergence:

$$|x_n - x| \leq K|x_0 - x|^{2^n}.$$

That is,

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

Remark: Remember that there's a difference between **cost** and **precision** for big-O notation. We say $W(\delta)$ "work" for cost and δ for precision.

So in our example, $\delta := \theta^n$, we have $n|\log\theta| = |\log\delta|$. Then we'd have:

$$\begin{aligned} W(\delta) = n &= O(|\log\delta|) \\ n &= \log\delta/\log\theta \\ &= 52 \end{aligned}$$

Out in the real world, how do we get K and θ?
We estimate θ by:

$$\begin{aligned} \frac{K\theta^{n+1}}{K\theta^n}\frac{|x_{n+1}-x|}{|x_n-x|} &= \frac{|x_{n+1}-x_n+x_n-x|}{|x_n-x_{n-1}+x_{n-1}-x|} \\ \theta &\approx \frac{|x_{n+1}-x_n|+|x_n-x|}{|x_n-x_{n-1}|+|x_{n-1}-x|} \\ &= \frac{K\theta^n+\delta_{n+1}}{K\theta^{n-1}+\delta_n} \\ &= \theta\left(\frac{K\theta^{n-1}+\delta_{n+1}/\theta}{K\theta^{n-1}+\delta_n}\right) \end{aligned}$$

So conclude:

$$\begin{aligned} 1 &= \frac{K\theta^{n-1}+\delta_n+(\delta_{n+1}/\theta-\delta_n)}{K\theta^{n-1}+\delta_n} \\ \implies 0 &= \delta_{n+1}/\theta-\delta_n \\ \theta &= \frac{\delta_{n+1}}{\delta_n} \end{aligned}$$

Remark: Strain gives an idea that for `fsolve( ..., x0)`, we can feed our found `x0` as the initial guess for `fsolve`.
For question 6, coding Newton's, we'll want a **diverse selection** of starting points that lead to different behavior.

2 Polynomial Interpolation

Consider a polynomial

$$P(t) = a_0 + a_1 t + \cdots + a_n t^n.$$

Why polynomials? They're easy to work with, and the parameters occur linearly, and they have a massive algebraic structure. However, for numerical analysis, we usually want to represent then in bases that are NOT the standard monomials.
In our first definition with the standard monomial basis, our polynomial lives (is centered) at 0, so we call it "zero-centric". So we can build another polynomial as follows, for $t_0 \neq t_1 \neq \cdots \neq t_n$:

$$P(t) = b_0 + b_1(t-t_0) + b_2(t-t_0)(t-t_1) + \cdots + b_n(t-t_0)(t-t_1)\cdots(t-tn-1)$$

We stop at $n-1$ index on the last term to ensure our polynomial is degree n. Consider that all of our terms in the second definition vanish at $t := t_0$,

besides b_0. And likewise, the others for other t_i. So this gives us some "triangular" representation of the polynomial.
The general goal is to approximate an arbitrary function $f(t)$ by some polynomial $P(t)$, and then we perform some operation that we want(ed) to do to f, but we do that instead to $P(t)$. For example, differentiating or integrating; if we don't know how to do this for f, we do it on the $P(t)$.
That is,

$$f'(t) \approx p'(t); \qquad \int_a^b f(t)\ dt \approx \int_a^b p(t)\ dt.$$

And of course, we can do this with other basis functions, such as trigonometric functions (as in Fourier analysis), but it turns out to be more or less the same thing.
Hence, interpolation is the **type** of approximation (which is perhaps the most famous) but not the only one. That is, we build some polynomial that matches $P(x) = f(x)$ at some set of points $x_0, \ldots, x_n$. This gives entirely different systems of equations, depending on what basis we choose.

$$\begin{aligned} a_0 + a_1 t_0 + a_2 t_0^2 + \cdots + a_n t_0^n &= f(t_0) \\ &\vdots \\ a_0 + a_1 t_n + a_2 t_n^2 + \cdots + a_n t_n^n &= f(t_n) \end{aligned}$$

This gives a $(n+1) \times (n+1)$ system of linear equations

$$\begin{bmatrix} 1 & t_0 & \cdots & t_0^n \\ \vdots & \ddots & & \vdots \\ 1 & t_n & \cdots & t_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}$$

where our conditions of solvability of this equation depends on this left matrix. So our left matrix is invertible IF AND ONLY IF $t_0 \neq t_1 \neq \cdots \neq t_n$, as known by the determinant.
An alternative to proving our matrix is invertible (determinant nonzero) is to show that the nullspace is nontrivial. Or alternatively, we can use:

$$1, (t - t_0), (t - t_0)(t - t_1), \ldots$$

This is known as the Newton basis, which gives us the triangular pattern as we saw before, where terms vanish for certain inputs. To see this explicitly, consider:

$$\begin{aligned} p(t_0) &= b_0 = f_0 \\ p(t_1) &= b_0 + b_1(t_1 - t_0) = f_1 \\ p(t_2) &= b_0 + b_1(t_2 - t_0)(t_2 - t_0)(t_2 - t_1) = f_2 \\ &\vdots \end{aligned}$$

So this implies there is a polynomial $p(t)$ interpolating $f(t)$ at $n+1$ points $t_0, \ldots, t_n$. This gives a lower-diagonal matrix, where if our main diagonal entries are nonzero, the matrix is invertible.

Recall from linear algebra, transformations for change-of-basis. A particularly useful result is:

$$\det B = \det(X)\det(A)\frac{1}{\det(X)},$$

where the determinant is **invariant** under the change-of-basis transformation. To conclude, we express the same requirements, but just in a different basis (now Newton basis).
To put this concretely, for example:

```
n = 0, constant
  p(t) = f(t_0)
n = 1 linear
  p(t0) = f(t0)
  p(t1) = f(t1)
```

Consider:

$$\begin{bmatrix}1 & t_0\\ 1 & t_1\end{bmatrix}\begin{bmatrix}a_0\\ a_1\end{bmatrix} = \begin{bmatrix}f_0\\ f_1\end{bmatrix}$$

Let $p(t) = b_0 + b_1(t - t_0)$. We have:

$$\begin{bmatrix}1 & 0\\ 1 & (t_1 - t_0)\end{bmatrix}\begin{bmatrix}b_0\\ b_1\end{bmatrix} = \begin{bmatrix}f_0\\ f_1\end{bmatrix}$$

This gives $b_0 = f_0$ and

$$\begin{aligned} b_0 + (t_1 - t_0)b_1 &= f_1\\ (t_1 - t_0)b_1 &= f_1 - f_0\\ b_1 &= \frac{f_1 - f_0}{t_1 - t_0}\end{aligned}$$

Hence we conclude:

$$\begin{aligned} p(t) = b_0 + b_1(t - t_0) &= f_0 + \frac{f_1 - f_0}{t_1 - t_0}(t - t_0)\\ &= \frac{f_0(t_1 - t_0) + (f_1 - f_0)(t - t_0)}{t_1 - t_0}\\ &= f_0\frac{t_1 - t}{t_1 - t_0} + f_1\frac{t_0 - t}{t_0 - t},\end{aligned}$$

which is known as the Lagrange form (Lagrangian Interpolation). In the above, we define:

$$\text{divided difference} := \frac{f_1 - f_0}{t_1 - t_0}.$$

Lagrange discovered the (fairly obvious) generalization for a "Lagrange ba-

sis" of degree n:

$$\begin{aligned}
L_0(t) &:= \frac{(t-t_1)(t-t_2)\cdots(t-t_n)}{(t_0-t_1)(t_0-t_2)\cdots(t_0-t_n)} \\
L_1(t) &:= \frac{(t-t_0)(t-t_2)\cdots(t-t_n)}{(t_1-t_0)(t_1-t_2)\cdots(t_1-t_n)} \\
&\vdots \\
L_j(t) &= \frac{\Pi_{k\neq j}(t-t_k)}{\Pi_{k\neq j}(t_j-t_k)} \\
&\vdots \\
L_n(t) &:= \frac{(t-t_1)(t-t_2)\cdots(t-t_{n-1})}{(t_n-t_1)(t_n-t_2)\cdots(t_n-t_{n-1})}
\end{aligned}$$

So we conclude, in the Lagrange basis,

$$\begin{aligned}
p(t) &= f_0L_0(t) + f_1L_1(t) + \cdots + f_nL_n(t) \\
&= \sum_{j=0}^{n} \left[f_jL_j(t)\right], \qquad L_j := \frac{\Pi_{k\neq j}(t-t_k)}{\Pi_{k\neq j}(t_j-t_k)}.
\end{aligned}$$

If we define $\omega_j := \frac{1}{\Pi_{k\neq j}(t_j-t_k)}$, we can write:

$$\begin{aligned}
p(t) &= \sum_{j=0}^{n} f_j\omega_j \frac{\Pi_k(t-t_k)}{t-t_j} \\
&= \omega(t)\sum_{j=0}^{n} \frac{f_j\omega_j}{t-t_j}.
\end{aligned}$$

This is called the **Barycentric form**, and it has its uses.
To continue our example earlier, we can write:

$$f_0\left(\frac{t_1-t}{t_1-t_0}\right) + f_1\left(\frac{t_0-t}{t_0-t_1}\right) = \underbrace{(t_0-t)(t_1-t)}_{\omega(t)}\left[\frac{f_0}{t_1-t_0}\frac{1}{t_0-t} + \frac{f_1}{t_0-t_1}\frac{1}{t_1-t}\right].$$

3 Error in Polynomial Interpolation

For example, if we interpolate with $n = 0$ (constant function f) then we have:

$$f(t) - p(t) = f(t) - f(t_0) = f'(\xi)(t-t_0),$$

by the mean value theorem. This error estimate (formula) is exactly the error for n-degree interpolation. Consider that $(t-t_0)$ makes the error equal to 0 at $t = t_0$. Also, $f'(\xi)$ is all about f, so if f is a constant, then the derivative is zero, and thus the error is identically zero across.
Then for linear interpolation $n = 1$:

$$f(t) - p(t) = \underbrace{[c\cdot f''(\xi)]}_{0 \text{ when } f \text{ is linear}} \cdot(t-t_0)(t-t_1)$$

Or equivalently, having two distinct points gives a unique line. To find the constant c, we test it on the function on the right, $\omega(t) := (t - t_0)(t - t_1)$. Hence

$$\begin{aligned} f(t) &= \omega(t) \\ f(t_0) &= f(t_0) = 0 \\ \implies p(t) &= 0, \end{aligned}$$

the linear polynomial going through points at zero, the zero function. Then equivalently write

$$\begin{aligned} \omega(t) &= t^2 + \cdots + \cdots + 1 \\ \omega''(t) &= 2! \end{aligned}$$

And we conclude $c := \frac{1}{2!}$ for above. Hence the error for **linear** interpolation is precisely:

$$f(t) - p(t) = \frac{1}{2!} f''(\xi) \left[(t - t_0)(t - t_1) \right]$$

So we want to sample close to the evaluation point to minimize the right side bracketed expression.

Lecture ends here.

Next time we'll see what goes wrong with interpolation.

Math 128A, Summer 2019
Lecture 10, Tuesday 7/9/2019

Topics Today:
- Deriving Interpolation Error
- Runge Phenomenon
- Chebyshev Points (as a Polynomial)

1 Review: Error in Polynomial Interpolation

Recall that for degree n, we take $(n+1)$ distinct points at which to interpolate (match values) between $p(x)$ and $f(x)$. We want to essentially predict the behavior (or values) of f by knowing other values of $p(x_i) = f(x_i)$.

$$\begin{aligned} p(t) &= a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \\ &= b_0 + b_1(t - t_0) + b_2(t - t_0)(t - t_1) + \cdots + b_n(t - t_0)\cdots(t - t_{n-1}), \end{aligned}$$

where this second (Newton) basis gives some triangular system of equations. This is good because we can look at them and immediately know if we have a unique solution. Or, we can write it via Lagrange as:

$$p(t) = f_0 L_0(t) + \cdots + f_n L_n(t),$$

and for **square** linear systems $(n + 1) \times (n + 1)$, if we have a solution, we are guaranteed (for free) that it is unique.
(That is, for a given function and a set of points at which to interpolate, regardless our interpolation basis or algorithm, our interpolating polynomial is unique. Hence we deduce the error from interpolation should be unique.)
Recall:

$$L_j(t) = \prod_{k \neq j} \left[\frac{t - t_k}{t_j - t_k} \right]$$

$$L_j(t_k) = \delta_j, k = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$$

And hence:

$$L_j(t) = \lambda_j \omega(t) \frac{1}{t - t_j}$$

$$\lambda_j = \Pi_{k \neq j} \frac{1}{t_j - t_k}$$

$$\omega(t) = (t - t_0)(t - t_1)\cdots(t - t_n)$$

And so $\omega(t)$ is a polynomial with zeros precisely at $t_0, t_1, \ldots, t_n$. This polynomial tells us if our distribution (spread) of distinct points is good for our purposes. We want to make the values of $|\omega(t)|$ small-valued for the intervals from which we are estimating.
Our error is zero at each of the interpolation points, and our $\omega(t)$ is also zero at each of the interpolation points. Hence:

Polynomial Interpolation Error:

$$f(t) - p(t) = \left(\underbrace{\frac{1}{(n+1)!}}_{\text{normalizes when } f(t)=\omega(t)} \right) \cdot \left(\underbrace{f^{(n+1)}(\xi)}_{0, \text{ if } f \text{ deg } n \text{ poly}} \right) \cdot \left(\underbrace{\omega(t)}_{0 \text{ at interpolating points}} \right).$$

The middle part is just to make sure that interpolating an n-degree f using $n+1$ polynomial p would cause the error to be exactly zero, because the polynomial will be unique (and exactly equal to f everywhere).
Remember that when taking the derivatives of $\omega(t)$, there's chain rule, but only one term counts.
We can prove this error formula, but for now we just 'construct' a polynomial that works for the interpolation error.
So, for linear interpolation, we use the above:

$$f(t) - \left[f(t_0) + \frac{f(t_1) - f(t_0)}{t_1 - t_0}(t - t_0) \right] = \frac{1}{2!} f''(\xi)(t - t_0)(t - t_1)$$

Example: $f(t) := e^t$ for $t = 0, 1, 2$. Then $f(0) = 1, f(1) = e, f(2) = e^2$. We'll use the Lagrange basis polynomial:

$$p(t) := \frac{(t-1)(t-2)}{(0-1)(0-2)} \cdot 1 + \frac{(t-0)(t-2)}{(1-0)(1-2)} \cdot e + \frac{(t-0)(t-1)}{(2-0)(2-1)} e^2$$

We can multiply this our and give it in the monomial basis, but this is more than sufficient. Strain says to just write out the Lagrange form, and you're good to go.
Then the error will be:

$$f(t) - p(t) = \frac{1}{3!} \cdot f^{(3)}(\xi) \cdot [(t-0)(t-1)(t-2)]$$

And we say: $e^{\xi} \in \left[e^0, e^2\right]$. So a worst-case a bound for our error can be:

$$|f(t) - p(t)| \leq \frac{1}{6} e^2 \max\{(t)(t-1)(t-2)\}$$

To find the maximum, consider:

$$\begin{aligned} g(t) &= t(t-1)(t-2) \\ g'(t) &= 0 \end{aligned}$$

So we should look at how the choice of interpolation points affects the error bounds $|\omega(t)|$.
We want to find the smallest $\max |\omega(t)| = |(t - t_0) \cdots (t - t_n)|$. To make it simple, just take some finite interval:

$$a \leq t_j \leq b.$$

We take $n+1$ intervals and place them on this interval $[a, b]$. And of course, shrinking the interval between the interpolation points makes the maximum

smaller. A bad way to distribute points is equispaced (but sometimes this is all we can do, because the data is given at equispaced points, like stock market data at every interval of time). So what is the best we can do, if we are able to decide how to distribute the points?
Consider: $t_0 = 0, t_1 = 1, \ldots, t_n = n$. Then for $0 \le t \le \frac{1}{2}$,

$$\omega(t) = (t-0)(t-1)(t-2)\cdots(t-n) \approx n!.$$

And similarly, for $\frac{n}{2} \le t \le \frac{n+1}{2}$,

$$\omega(t) \approx \left(\left(\frac{n}{2}\right)!\right)^2.$$

To view this, consider from Stirling's:

$$\begin{aligned} n! &\approx \left(\frac{n}{e}\right)^n \\ \left[\left(\frac{n}{2}\right)!\right]^2 \approx \left[\left(\frac{n}{2e}\right)^{n/2}\right]^2 &= \left(\frac{n}{2e}\right)^n \omega(t) \qquad \approx 2^{-n} \cdot n! \end{aligned}$$

So for $n = 20$, the error bound is a million (2^{20}) times larger near the end points than in the center of the interval. We call this the "Runge Phenomenon", (pronounced 'roon-huh').

Runge Phenomenon - Mantra:
High-degree polynomial interpolation at equispaced points is **wildly inaccurate** near the ends of the (interpolating) interval.

Consider that no one is making us do high-degree interpolation, so we can do piecewise interpolation with lower degree polynomials. And perhaps instead of polynomials, we may use our favorite functions (like fourier or splines). Or, instead of exact interpolations, we can just keep a least-squares fit near (but not exact) to the polynomial. We can also use Chebyshev points to get around equispaced points. Or, we can just interpolate on a larger interval but only pull (test) on the middle of the region, where our error is low.

2 Chebyshev Points

These are very simple and completely eliminate the Runge Phenomenon. The central idea is to build $\omega(t)$ which has $n+1$ zeroes on $[-1, 1]$ (convention), where we make all the maxima and minima the same height, $\pm$. We call this "equioscillating".
We simply construct polynomials T that guarantee our desired condition that $|\sup T| = |\inf T|$. That is, we want to bound the polynomial function min and max values to precisely hit $[-1, 1]$.

$T_0(t) := 1$ (interpolating (1,1))
$T_1(t) := t$ (interpolating (-1,-1) and (1,1))
(now interpolating (-1,1), (0,-1), (1,1)):

$$\begin{aligned} T_2(t) &:= 1\frac{(t-0)(t-1)}{(-1-0)(-1-1)} + (-1)\frac{(t-1)(t-(-1))}{(0-1)(0-(-1))} + 1\frac{(t-(-1))(t-0)}{(1-(-1))(1-0)} \\ &= 2t^2 - 1 = 2t \cdot T_1(t) - T_0(t) \end{aligned}$$

But equivalently, defining $\cos x := t; x := \cos^{-1}(t)$, this is:

$$T_2(t) := \cos(2t) = 2\cos^2(x) - 1,$$

and we easily check our definition works for T_1, T_0 as above. Generalizing, we have as a candidate for an equioscillating polynomial:

$$T_k(t) = \cos(kx).$$

So we say: $|t| \leq 1 \iff 0 \leq x \leq \pi$. Now the only 'scary' part is that we don't know this is a polynomial (it certainly doesn't look like it).
Recall:

$$\begin{aligned}
\cos(a+b) &= \cos a \cos b - \sin a \sin b \\
\cos(a-b) &= \cos a \cos b + \sin a \sin b \\
\cos(a+b) + \cos(a-b) &= 2\cos b \cos a \quad \text{(adding the above equations)}
\end{aligned}$$

And letting $a := bx, b := x$, we get the following recurrence relation:

$$T_{k+1}(t) + T_{k-1} = 2t\ T_k(t).$$

Or in a more useful form:

$$T_{k+1} = 2t\ T_k - T_{k-1}$$

And so,

$$\begin{aligned}
T_0 &= 1, \qquad T_1 = t, \\
T_2 &= 2t(t) - 1 \\
T_3 &= 2t\ (2t^2 - 1) - t \\
&= 4t^3 - 3t \implies t = \left\{0, \frac{\sqrt{3}}{2} \frac{-\sqrt{3}}{2}\right\}
\end{aligned}$$

And we can keep going onwards to generate polynomials.
A conjecture can be that T_k is even iff k is even, and similarly T_k is odd if k is odd. Consider the following definition:

$$\begin{aligned}
T_k(t) &= \cos(kx) = 0 \\
kx &= (2j+1)\pi/2, \qquad j = 0, 1, \ldots, (k-1) \\
\implies x &= \frac{(2j+1)\pi}{2k}
\end{aligned}$$

where this is a polynomial in disguise. And we write:

$$t_j = \cos\left(\frac{2j-1}{2k}\pi\right); \qquad 1 \leq j \leq k$$

so these guys are numbered backwards, from right to left, which is ok (because we could add interpolation points in any order).
So effectively, we took a circle, divide the angle of 0 to π evenly, and project the points on the circle down onto the x-axis.

Remark: Using Chebyshev points is a good way to distribute points when we don't know the function we are interpolating.

From the definition of our recurrence relation

$$T_{k+1} := 2t\ T_k - T_{k-1}; \qquad T_0 := 1, T_1 := t$$

we can see that T_k must be a k-degree polynomial. We look for the characteristic equation (which happens because we look for T_k as a linear combination of two powers r^k, the roots of the characteristic equation). Then:

$$\begin{aligned} T_k &= r^k \\ r^{k+1} &= (2t)r^k - r^{k-1} \quad \text{(from the recurrence relation)} \\ r^2 + 2tr + 1 &= 0 \quad \text{(dividing by } r^{k-1}\text{)} \\ \implies r &= t \pm \sqrt{t^2-1} = t \pm i\sqrt{1-t^2}. \end{aligned}$$

Because we have two roots, we then have:

$$T_k = ar_+^k + br_-^k$$

And we set:

$$\begin{aligned} T_0 &= 1 = a + b \\ T_1 &= ar_+ + br_- = t \end{aligned}$$

Solving this our favorite way (i.e. 2×2 matrix), we have:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{-2\sqrt{t^2-1}} \begin{bmatrix} r_- - t \\ -r_+ + t \end{bmatrix}$$

where:

$$\begin{aligned} r_- = t - \sqrt{t^2-1} \quad &\implies \quad r_- - t = -\sqrt{t^2-1} \\ r_+ = t + \sqrt{t^2-1} \quad &\implies \quad r_+ - t = -\sqrt{t^2-1} \end{aligned}$$

And we conclude:

$$\begin{aligned} T_k(t) &= \frac{1}{2}\left[\left(t+\sqrt{t^2-1}\right)^k + \left(t-\sqrt{t^2-1}\right)^k\right] \\ &= \frac{1}{2}\left[e^{ikx} + e^{-ikx}\right] = \cos(kx), \end{aligned}$$

and we have that cosine is precisely a polynomial satisfying our requirements for T_k (recall we set $t := \cos(kx)$).

Lecture ends here.

Math 128A, Summer 2019
Lecture 11, Wednesday 7/10/2019

Topics today:
- Newton Interpolation
- Divided Differences
- Hermite Interpolation

Midterm topics (first 3 weeks): IEEE arithmetic, Bisection, Rate of Convergence, Iteration (fixed point, newton), Interpolation (lagrange + **error**, newton, hermite)

Recall we have Lagrange Interpolation, Newton Interpolation, Hermite Interpolation.

1 Newton Interpolation

The idea here is to use the Newton basis (we just call it this because we use this basis for Newton's):

$$1, t - t_0, (t - t_0)(t - t_1), \ldots, (t - t_0) \cdots (t - t_n)$$

And recall, if our diagonal entries of this triangular matrix are nonzero, then we have a unique solution. And our polynomial is:

$$p(t) = a_0 + a_1(t - t_0) + \cdots + a_n(t - t_0) \ldots (t - t_n)$$

This basis is 'simpler' than the Lagrange basis (where each term is of degree n and vanishes at each interpolation point). So in a Newton basis, we build an interpolating polynomial **recursively** by:

$$p_n(t) := \overbrace{p_{n-1}(t)}^{\deg\, n-1} + a_n \left[(t - t_0)(t - t_1)(t0 - t_{n-1})\right]$$

where $p_n(t)$ is the polynomial evaluating over $t_0, \ldots, t_n$, and $p_{n-1}(t)$ interpolates over $t_0, \ldots, t_{n-1}$. The next term $a_n [\cdots]$ has to raise the degree by 1 but **'not mess with'** the already interpolated points from the left term. Thus the bracketed expression ensures that, and a_n just has to be chosen to interpolate the new point t_n. Hence:

$$\begin{aligned} p_n(t_n) &:= p_{n-1}(t_n) + a_n \left[(t_n - t_0)(t_n - t_1) \cdots (t_n - t_{n-1})\right] \\ &= f_n, \end{aligned}$$

and we simply solve this equation to find a_n:

$$a_n = \frac{f_n - p_{n-1}(t_n)}{(t_n - t_0)(t_n - t_1) \cdots (t_n - t_{n-1})}$$

Notice that $p_{n-1}(t_n)$ is also an approximation to f_n, so we guess that the numerator will be tiny, and the denominator should be tiny (if we designed the distribution well).
However, there's a better formula:

Definition: Divided Difference -

$$\begin{aligned}
a_1 &:= \frac{f[t_1] - f[t_0]}{t_1 - t_0} \\
&= \frac{f_1 - f_0}{t_1 - t_0} = f[t_0, t_1] = f[t_1, t_0] \\
a_2 &:= \frac{f[t_1, t_2] - f[t_0, t_1]}{t_2 - t_0} \\
&= f[t_0, t_1, t_2] = f[t_2, t_1, t_0] = f[t_2, t_0, t_1] = \cdots \\
&\vdots \\
a_n &:= \frac{f_n - p_{n-1}(t_n)}{(t_n - t_0)(t_n - t_1)\cdots(t_n - t_{n-1})} \\
&= f[t_0, t_1, \ldots, t_n] = \text{or any permutation of } t_i
\end{aligned}$$

Example: Divided Difference (to generate the above):

$$\begin{aligned}
n &= 0; \\
p_0(t) &= f(t_0) = f[t_0] \\
n &= 1; \\
p_1(t) &= p_0(t) + a_1(t - t_0) \\
p_1(t_1) &= f_1 \\
&\vdots
\end{aligned}$$

Example: Recall the function $f(t) := e^t$. Let us define: $t_0 := 0, t_1 := 1, t_2 := 2$. Then

$$\begin{aligned}
f[0] &= f[t_0] = 1, \\
f[1] &= f[t_1] = e, \\
f[2] &= f[t_2] = e^2
\end{aligned}$$

Our divided difference table looks like:

$$\begin{array}{lll}
0 & f_0 = 1 & \dfrac{e-1}{1-0} = e - 1 \qquad \dfrac{e^2 - e - (e-1)}{2-0} = \dfrac{e^2 - 2e + 1}{2} \\
1 & f_1 = e & \dfrac{e^2 - e}{2 - 1} = e^2 - e \\
2 & f_2 = e^2 &
\end{array}$$

So this readily gives us the interpolating polynomial in the Newton basis:

$$p(t) = \underbrace{1}_{p(0)=1} + \underbrace{(e-1)}_{p(1)=e}(t-0) + \frac{e^2 - 2e + 1}{2}(t-0)(t-1)$$

and we can 'check' values easily in the Newton basis. Compare this to Lagrange from yesterday:

$$p(t) = 1\frac{(t-1)(t-2)}{(0-1)(0-2)} + e\frac{(t-0)(t-2)}{(1-0)(1-2)} + e^2\frac{(t-0)(t-1)}{(2-0)(2-1)}$$

Later, we'll talk about Hermite interpolation, which is a generalization of a Taylor expansion which expands through derivative orders, and newton/lagrange is interpolation at a particular degree. We think of these as limiting or edge cases of Hermite interpolation which involves interpolation at a particular derivative order.

2 Error in Newton Interpolation

Recall:

$$f(t) - p_n(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\left[(t-t_0)(t-t_1)\cdots(t-t_n)\right]$$

Consider:

$$p_{n+1}(s) = p_n(s) = f[t_0, \ldots, t_n, \underbrace{\quad}_{t}]\omega(t),$$

so that $p_{n+1}(t) = f(t)$ for a fixed t. We have the interpolation error via the top expression. Then we fix t and interpolate at another point, that point being t. Then when we evaluate the polynomial, we have:

$$p_{n+1}(t) = f(t) = p_n(t) + f[t_0, \ldots, t_n, t]\omega(t).$$

Thus we conclude these must be the same error (formula), and we have the following theorem:

Theorem 2.1.

$$f[t_0, t_1, \ldots, t_n, t] = \frac{f^{(n+1)}(\xi)}{(n+1)!},$$

where $\xi \in [\min\{t_j, t\}, \max\{t_j, t\}]$.

This equivalently states:

$$f[t_0, t_1, \ldots, t_n] = \frac{f^{(n)}(\xi)}{(n)!},$$

which is not surprising because this gives: $f[t_0] = f(t_0)$ and

$$f[t_0, t_1]\frac{f(t_1) - f(t_0)}{t_1 - t_0} = \frac{f'(\xi)}{1!},$$

which is equivalent to the MVT for $n = 1$. For higher orders, we get something more interesting.
Suppose $t_1 \to t_0$. Then

$$f[t_0, t_1] = \frac{f'(\xi)}{1!} \to f'(t_0)$$

Equivalently, for all $t_j \to x$,

$$f[t_0, t_1, t_2] = \frac{f''(\xi)}{2!} \to \frac{f''(x)}{2!}$$

This allows us to do **Hermite interpolation**, by interpolating using the derivative **at the same point** at different derivatives.
If we set all $t_j \to t_0$,

$$\begin{aligned} p_n(t) &= f[t_0] + f[t_0, t_1](t - t_0) \\ &\quad + \cdots \\ &\quad + f[t_0, \ldots, t_n](t - t_0) \cdots (t - t_{n-1}) \\ &\to f(t_0) + f'(t_0)(t - t_0) + \cdots + \frac{f^{(n)}(t_0)}{n!}(t - t_0)^n, \end{aligned}$$

which is identical to Taylor's expansion.

Example: Hermite Interpolation We set:

$$p(t_0) = f_0, \quad p'(t_0) = f_0', \quad p(t_1) = f_1, \quad p'(t_1) = f_1'.$$

We create a divided-difference table: We take the top row of our divided difference table, which was:

$$0 \qquad 1 \qquad 1 \qquad e-2 \qquad 3-e \qquad \frac{e^2 - e + 2}{2} \qquad \frac{6e - 3e^2 - 1}{4},$$

and we set:

$$\begin{aligned} p(t) &= 1 + 1\,(t - 0) + (e - 2)\,(t - 0)^2 \\ &\quad + (3 - e)(t - 0)^2(t - 1) \\ &\quad + \frac{e^2 - e + 2}{2}(t - 0)^2(t - 1)^2 \\ &\quad + \frac{6e - 3e^2 - 1}{4}(t - 0)^2(t - 1)^2(t - 2), \end{aligned}$$

which gives us our desired quintic Hermite polynomial, interpolating at 3 distinct points.

2.1 Why or When Would We Use Hermite Interpolation?

Taking equispaced points, if we let $n = 2k$, suppose we instead take two points t_0, t_1 centered in the larger interval, but stacked the derivatives at the center (vertically) rather than spread out on the interval. For the same amount of 'work $n = 2k$,' we compare the errors:

$$\begin{aligned} \omega(t) &= (t - t_0) \cdots (t - t_n) \\ \omega(t) &= (t - t_0)^k (t - t_1) k, \end{aligned}$$

and the second polynomial (hermite with derivatives) has a lot less error, for the same amount of work. Also notice that hermite interpolation with fourier can be good, because evaluating derivatives of the fourier (trig) functions is free, just like evaluating the fourier function itself.

Error for Hermite Interpolation : Consider:

$$\begin{matrix} f(t_0) & & f'(t_0) \\ \vdots & & \vdots \\ f(t_n) & & f'(t_n) \end{matrix}$$

So the error is going to be the same as Lagrange for this polynomial degree, $\deg p = 2n+1$. We inherit the Lagrange error and let points merge:

$$f(t) - p(t) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(t-t_0)^2 \cdots (t-t_n)^2$$

We call this strategy 'confluence', as to let the points flow together.
Notice that we have squared factors on the right because we need to make the function *and* the derivative vanish at that point. This is equivalent to having, for example, $t_0 + \epsilon$ and letting $\epsilon \to 0$.

3 Lagrange Basis for Hermite Interpolation

Let

$$p(t) := \sum_{j=0}^{n} f_j \underbrace{A_j(t)}_{\deg\ 2n+1} + \sum_{j=0}^{n} f_j' B_j(t)$$

where

$$\begin{aligned} A_j(t_k) &= \delta_{j,k} & \qquad B_j(t_k) &= 0 \\ A_j'(t_k) &= 0 & \qquad B_j'(t_k) &= \delta_{j,k}. \end{aligned}$$

So first, we try to be clever (efficient) to satisfy the above conditions:

$$\begin{aligned} \underbrace{A_j(t)}_{\deg\ 2n+1} &= \underbrace{(a + b(t-t_j))}_{\text{to raise deg}} \cdot \underbrace{L_j(t)^2}_{\deg\ 2n} \\ B_j(t) &= (t-t_j)L_j(t)^2 \\ B_j'(t) &= L_j(t)^2 + (t-t_j)2L_j(t)L_j'(t) \end{aligned}$$

And for A_j and A_j',

$$\begin{aligned} A_j(t_k) &= (1 + b(t_k - t_j))\, L_j(t_k)^2 \\ A_j'(t) &= bL_j(t)^2 + 2\,(1 + b(t-t_j))\, L_j'(t)L_j(t) \\ A_j'(t_j) &= b + 2L_j'(t_j) \\ b &= -2L_j'(t_j), \end{aligned}$$

and to summarize,

$$\begin{aligned} A_j(t) &= \left[1 - 2L_j'(t_j)(t-t_j)\right] L_j(t)^2 \\ B_j(t) &= (t-t_j)L_j(t)^2. \end{aligned}$$

Math 128A: Polynomial Interpolation

Nate Armstrong

Lecture 7/10/19

Midterm Info

2.3 Review

We want to solve the problem below.

$$p(t_j) = f_j \quad 0 \leq j \leq n$$

where p is a degree n polynomial. We have multiple formulas for the above. We could use Lagrange interpolation, which is

$$p(t) = \sum_{j=0}^{n} f_j L_j(t)$$

where $L_j(t)$ is the Lagrange basis function for t_j. We can also use Newton interpolation, which is where

$$p(t) = \sum_{j=0}^{n} f[t_0, t_1, \ldots, t_j](t - t_0) \cdots (t - t_{j-1})$$

Generally, this is solved by using a table and building the differences recursively. Finally, if we want to solve this problem and also approximate the derivatives of f_j at these points, we have $(n+1)(m+1)$ conditions on our polynomial, where m is the number of derivatives we want. Note that if $m = 0$, this is the same as regular polynomial interpolation. The degree of p is then $(n+1)(m+1) - 1$.

This is called Hermite interpolation.

Remark
Most people call general interpolation 'Lagrange Interpolation'. This is a special case of Hermite Interpolation. Hermite interpolation can handle different amounts of derivatives in interpolation, but you cannot handle derivatives at a point without the function itself, or any gaps in the sequence of derivatives. In those cases, it is called Birkhoff Interpolation. It is not known when this is possible.

It is very fast to change data points when the interpolated polynomial is in Lagrange form. However, when the data changes in Newton form, you have to do $O(n^2)$ work to recreate the polynomial. However, if the data is fixed and t is changing, Newton's form is much more efficient (around $3n$ operations compared to $O(n^2)$). There are other ways of evaluating interpolating polynomials which can let you do much less computational work, if you're willing to do more thinking about it.

Hermite interpolation can use both Lagrange and Newton form. However, doing it in Newton form is much easier, as finding the Lagrange basis functions can be a huge pain.

Example
Suppose I have f_0, f_0', f_0'', f_1. Then, I interpolate as

$$\begin{array}{c|c|c|c|c} 0 & f_0 & f_0' & \frac{f_0''}{2!} & f[t_0,t_0,t_0,t_1] \\ 0 & f_0 & f_0' & \frac{f[t_0,t_1]-f_0'}{1-0} = f[t_0,t_0,t_1] & \\ 0 & f_0 & \frac{f_1-f_0}{1-0} & & \\ 1 & f_1 & & & \end{array}$$

Thus, we get a final polynomial of

$$p(t) = f[t_0] + f[t_0,t_0](t-t_0) + f[t_0,t_0,t_0](t-t_0)^2 + f[t_0,t_0,t_0,t_1](t-t_0)^3$$

Note from Strain: We don't know how to solve using the table if we don't put matching values next to each other while solving the table.

Last time, we also computed the Lagrange basis of the Hermite interpolation. We can find the Lagrange basis by Newton, using the equations that

$$\begin{aligned} A_j(t_k) &= \delta_{jk} & B_j(t_k) &= 0 \\ A_j'(t_k) &= 0 & B_j'(t_k) &= \delta_{jk} \end{aligned}$$

Then, I state that $p(t) = \sum_{j=0}^{n} f_j A_j(t) + f_j' B_j(t)$ Then, I can use Newton to find all the A_js and B_js for this particular case.

3 Numerical Differentiation

Although you can always compute the derivative of a function if you have a formula, most of the time you just have data points. You want to find the derivative at or near the data points. The usual way to do it is by interpolating with a polynomial, and then take the derivative at the point you are interested in.

1. Interpolate by $p(t)$ at $t_0, \ldots, t_n$.
2. Approximate $f'(a)$ by $p'(a)$, as you can compute $p'(a)$.

Example
Suppose I have $f(t_0)$, and $f(t_1)$. If I approximate with a constant polynomial, I will certainly get $p'(t) = 0$.

If I approximate with a linear polynomial, I will get $p'(t) = \frac{f(t_1)-f(t_0)}{t_1-t_0} = f[t_0, t_1]$. As this is a divided difference, we know it is equal to the first derivative of f' at some point. Unfortunately, this point is most likely not the point we want.

Now, we are interested in learning the error in such an approximation. We know that

$$f(t) - p(t) = \frac{f^{(n+1)}(\zeta)}{(n+1)!}\omega(t)$$

which we can differentiate to

$$f'(t) - p'(t) = \frac{f^{(n+1)}(\zeta)}{(n+1)!}\omega'(t) + \frac{1}{(n+1)!}\omega(t)\frac{d}{dt}f^{(n+1)}(\zeta)$$

The right side is ugly, however we know that $\omega(t_j) = 0$ for all j, and so if we only worry about the error at the interpolation points, we can ignore the right term. Thus, at $t = t_j$, the error is just

$$E_j = \frac{f^{(n+1)}(\zeta)}{(n+1)}\omega'(t_j)$$

and so we need to find $\omega'(t_j)$. Because I have $n+1$ terms of the form $(t-t_j)$, I have $n+1$ terms which each have 1 term missing (the one that was differentiated) via the chain rule. Formally,

$$\omega'(t_j) = \sum_{k=0}^{n}\prod_{l\neq k}(t_j - t_l)$$

However, as this is evaluated at some t_j, the only nonzero term is the one where $(t-t_j)$ was differentiated. Thus, at $t = t_j$, we get

$$\omega'(t) = \prod_{k\neq j}(t_j - t_k).$$

which is a relatively well-behaved product. In fact, this is a factor of $L_j(t)$. We can actually write that

$$L_j(t) = \frac{\prod_{k\neq j}(t-t_k)}{\prod_{k\neq j}(t_j-t_k)} = \frac{\omega(t)/(t-t_j)}{\omega'(t)} = \frac{\omega(t)}{\omega'(t)(t-t_j)}$$

This formula is weird, but very useful (according to prof).

3.1 Coefficients

In order to solve for the derivative of a polynomial which we have interpolated, we want the derivative in terms of the values of the function at a certain point.

We want a formula of the form

$$f^{(m)}(a) = \frac{\delta_{n0}^m f(t_0) + \cdots + \delta_{nm}^m f(t_n)}{h^m}$$

We construct general formulas using the Taylor expansion, using

$$f(a+h) = f(a) + hf'(a) + \frac{1}{2}h^2 f''(a) + \ldots$$

and that

$$f(a-h) = f(a) - hf'(a) + \frac{1}{2}h^2 f''(a) + \ldots$$

to get

$$f(a+h) - 2f(a) + f(a-h) = h^2 f''(a) + \frac{1}{6}h^3 f'''(\zeta_+) - \frac{1}{6}h^3 f'''(\zeta_-)$$

The error here is of $O(h^2 f^{(4)}$, as we can take one more term in the Taylor expansion and let the third terms cancel. In the case of the second derivative, we have $\delta_{2,0}^2 = 1$, $\delta_{2,1}^2 = -2$, and $\delta_{2,2}^2 = 1$. In fact, we need that all of the δs must sum to 0 when added. This is the case because if the interpolated polynomial is $f(x) = c$, it must correctly return 0.

If we interpolate the polynomial in Lagrange form, then we write

$$p^{(m)}(a) = \sum_{j=0}^{m} f_j L_j^{(m)}(a).$$

Thus, we want to compute $\delta_{nj}^m(a) = L_j^{(m)}(a)$.

The notation is complex, but $\delta_{nj}^m(a)$ means the mth derivative of the Lagrange basis function for n points, applied for the jth point, evaluated at a.

3.1.1 'Newton'ing the Lagrange

When Newton is a verb, it means adding the points one at a time. We let

$$L_{nj}(t) = \prod_{k \neq j} \frac{t - t_k}{t_j - t_k} = \frac{t - t_n}{t_j - t_n} L_{n-1,j}(t)$$

which is true for $j < n$. Now, to calculate the derivatives of this, I get

$$L'_{nj} = \frac{1}{t_j - t_n} L_{n-1,j}(t) + \frac{t - t_n}{t_j - t_n} L'_{n-1,j}(t)$$

Then, I look for L''_{nj}. This is

$$L''_{nj}(t) = \frac{2}{t_j - t_n} L'_{n-1,j}(t) + \frac{t - t_n}{t_j - t_n} L''_{n-1,j}(t).$$

Now, we apply this to the third derivative.

$$L'''_{nj}(t) = \frac{3}{t_j - t_n} L''_{n-1,j}(t) + \frac{t - t_n}{t_j - t_n} L'''_{n-1,j}(t).$$

This gives me a recurrence relation. I see that

$$\delta^m_{nj} = \frac{m}{t_j - t_n} \delta^{m-1}_{n-1,j} + \frac{a - t_n}{t_j - t_n} \delta^m_{n-1,j} \tag{1}$$

as a recurrence relation for $j < n$. The way to organize this would be to have a target n, then calculate this for all j and $m = 0$. This is pretty easy, as for $m = 0$ the above formula is trivial. Then, we calculate the values here for $m = 1$, using the formula

$$\delta^1_{nj} = \frac{1}{t_j - t_n} \delta^0_{n-1,j} + \frac{a - t_n}{t_j - t_n} \delta^1_{n-1,j}$$

It's hard to think about this problem.

We spent the rest of lecture trying to think of an ordering in which to program this. At the end, Strain mentioned that he took this time in order to show that the problem is not trivial. It seemed that we could first compute for $m = 0$, then use those values to compute the above. We know that $\delta^m_{00} = 0$ for any $m > 1$. We also solved for some small index values of δ, but I did not type those up.

Math 128A, Summer 2019
Lecture 13, Monday 7/15/2019

CLASS ANNOUNCEMENTS: Midterm on Wednesday, only 1 hour long, 3 problems. No (numerical) integration.
Goals today: Talk about Numerical integration, also known as Quadrature.

1 Homework 4 Review

1.1 Problem 1

Recall problem 1 asks us to interpolate

$$f(t) = \frac{1}{1+t^6}.$$

Consider the system of linear equations

$$\sum_{j=0}^{n}$$

In the solution I already built, we demonstrated that interpolation in the monomial basis is bad (θ^k). Consider instead:

1.2 Error in Hermite Interpolation

$$f(t) - p(t) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\left[\omega(t)\right]^2$$

We say that the error is bounded by $f^{(2n+2)}(\xi)$, which we (optionally, in 128B) express in norms:

$$\|f\|_\infty = \max_x |f(x)|.$$

2 Quadrature (Numerical Integration)

We call numerical integration sometimes as quadrature because in the old days, we would draw grids and count the squares for the 'area under the curve' for integration.
Today we'll talk about the **Trapezoidal rule**, **not to derive Newton-Cotes** as it is bad, but rather talk about **Endpoint correction**. That is, we look at two things: δ_{jk}^m and the Euler-Maclaurin summation formula.
The general idea of numerical integration:

$$\int_a^b f(x)\ dx = \int_a^b p(x)\ dx + \int_a^b E(x)\ dx,$$

where $p(x)$ is an approximating polynomial (something we know how to approximate), and $E(x)$ is the error.

For example, $p(x)$ interpolates $f(x)$ at $x_0, \dots, x_n$. This implies $p(x_j) = f(x_j)$ for $0 \le j \le n$.

$$\begin{aligned}\int_a^b p(x)\ dx &= \int_a^b f(x_0)L_0(x) + \cdots + f(x_n)L_n(x)\ dx \\ &= f(x_0)\underbrace{\int_a^b L_0(x)\ dx}_{w_0} + \cdots + f(x_n)\underbrace{\int_a^b L_n(x)\ dx}_{w_n} \\ &= w_0 f(x_0) + w_1 f(x_1) + \cdots + w_n f(x_n)\end{aligned}$$

So we take the Lagrange-basis . This is a once-in-a-lifetime calculation, as they do not depend on f. We call these the weights $w_0, \dots, w_n$.
Let's look at the stupidest possible example:

$$\begin{aligned}n = 0: \qquad p(x) &= f(x_0) \cdot 1 \\ L_0(x) &= 1 \\ \int_a^b L_0(x)\ dx &= b - a =: w_0\end{aligned}$$

Hence:

$$\int_a^b f(x)\ dx \approx (b-a)f(x_0).$$

Then for our error, we can write:

$$\int_a^b f(x)\ dx - [w_0 f(x_0) + \cdots + w_n f(x_n)] = \int_a^b \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\omega(x)\ dx$$

This is not a particularly useful error formula
If $\omega(x)$ **does not change sign** over our interval $[a, b]$, then we can use the MVT for integrals to estimate:

$$\text{error} = \frac{f^{(n+1)}(\varphi)}{(n+1)!}\int_a^b \omega(x)\ dx,$$

where $\varphi \in [a, b]$ is some unknown point.
Otherwise, we have:

$$\begin{aligned}|\text{error}| &\le \int_a^b \frac{|f^{(n+1)}(\xi(x))|}{(n+1)!}|\omega(x)|\ dx \\ &\le \frac{\max_x |f^{(n+1)}(x)|}{(n+1)!}\underbrace{\int_a^b |\omega(x)|}_{\text{constant}}\ dx\end{aligned}$$

where the second inequality follows directly from the triangle inequality.

Example : $n = 1$:

$$p(x) = \underbrace{\frac{x - x_1}{x_0 - x_1}}_{L_0(x)} f(x_0) + \underbrace{\frac{x - x_0}{x_1 - x_0}}_{L_1(x)} f(x_1)$$

Solution. Consider:

$$\int_a^b \frac{x - x_1}{x_0 - x_1}\ dx = \frac{1}{2}\frac{(b-x_1)^2 - (a-x_1)^2}{x_0 - x_1}$$

Then this gives:

$$\int_a^b f(x)\ dx = f(x_0)w_0 + f(x_1)w_1$$

We'll use the Trapezoidal Rule, in that if $x_0 := a$ and $x_1 := b$, the above gives:

$$w_0 = \frac{1}{2}\frac{-(a-b)^2}{a-b} = \frac{1}{2}(b-a) > 0 \text{if } b > a.$$

And we get w_1 for free; that is:

$$w_1 = \underbrace{w_0 + w_1}_{\approx \int_a^b 1\ dx} + w_0$$

And this is a constant integral, so we conclude:

$$\begin{aligned} w_0 + w_1 = w_0 \cdot 1 + w_1 \cdot 1 = \int_a^b 1\ dx = (b-a) \\ \implies w_1 = \frac{1}{2}(b-a) \end{aligned}$$

Usually we start off Trapezoidal Rule by writing:

$$\int_a^b f(t)\ dt = \frac{1}{2}f(0) + \frac{1}{2}f(1) + E,$$

where

$$\begin{aligned} E &= \int_0^1 \frac{f''(\xi)}{2!}(t-0)(t-1)\ dt \\ &= \frac{f''(\xi)}{2!}\int_0^1 (t)(t-1)\ dt \\ &= \frac{1}{2!}\cdot\left(\frac{1}{3} - \frac{1}{2}\right) f''(\xi) \\ &= \frac{-1}{12} f''(\xi) \end{aligned}$$

□

2.1 Compounding

We take the interval $[a, b]$ and break down into intervals:

$$\begin{aligned} [a,b] &= [a, a+h] \cup [a+h, a+2h] \cup \cdots \cup [b-h, b], \qquad h = \frac{b-a}{n} \\ &= \bigcup_{j=0}^{n-1} [a + jh, a + (j+1)h] \end{aligned}$$

and we get:

$$\int_a^b f(x)\ dx = \sum_{j=0}^{n-1} \left[\underbrace{\int_{a+jh}^{a+(j+1)h} f(x)\ dx} \right]$$

Runge says to not interpolate a function and use the entirety of the interval to perform integration with one high-degree interval and calculating weights across the entire interval. Instead, we 'derive' the obvious Trapezoidal Rule simply taking the linear interpolants of each small interval.
That is, for our previous example we have:

$$\begin{aligned}\int_a^b f(x)\ dx &= \sum_{j=0}^{n-1} \left[\underbrace{\int_{a+jh}^{a+(j+1)h} f(x)\ dx} \right] \\ &= \frac{1}{2}hf(a+jh) + \frac{1}{2}hf(a+(j+1)h),\end{aligned}$$

and

$$E = \frac{-h^3}{12} f''(\xi),$$

where the power of 3 follows from the dimension of $1 \to h$ changing

$$\frac{1}{2}f(0) + \frac{}{2}f(1) + E \to \frac{1}{2}f(0) + \frac{}{2}f(1) + E;$$

and hence our error E has dimensions

$$\frac{HF}{H^2},$$

and in order to compensate, we need to insert $1 \mapsto h^3$ in the error formula in changing from the $[0,1]$ interval to smaller intervals).
So in all, we have:

$$\begin{aligned}\int_a^b f(x)\ dx &= \sum_{j=0}^{n-1} \underbrace{\frac{1}{2}hf_j + \frac{1}{2}hf_{j+1}} - \frac{1}{12}\sum_{j=0}^{n-1} h^3 f''(\xi_j) \\ &\quad \frac{1}{2}h\left(\underbrace{f_0 + f_1} + \underbrace{f_1 + f_2} + \cdots\right) \\ &= h\left(\frac{1}{2}f_0 + f_1 + \cdots + f_{n-1} + \frac{1}{2}f_n\right) \quad \text{Trapezoidal Rule} \\ &\quad - \frac{h^3}{12}f''(\xi) \quad \text{Error term}\end{aligned}$$

Whatever oscillation we have can be blamed entirely on f and not our integration formula / algorithm, unlike in Newton-Cotes.
Define $f_j := f(a+jh)$. In summary, our Compound Trapezoidal Rule gives us:

$$\int_a^b f(x)\ dx = h\underbrace{\sum_{j=0}^{n} f_j} - \frac{h^2}{12}\frac{b-a}{n}\sum_{j=0}^{n-1} f''(\xi_j),$$

where the underbraced term is equal to $\frac{1}{2}$ on the first and last terms, and our sum on the right is an average (MVT). So we write:

$$\int_a^b f(x)\ dx = h\sum_{j=0}^{n} -\frac{h^2}{12}(b-a)f''(\xi)$$

The rate of convergence is $O(h^2) = O(\frac{1}{n^2}) = \varepsilon$. We say that the work required is

$$n = O(\varepsilon^{-1/2}) = O\left(\frac{1}{\sqrt{\varepsilon}}\right)$$

It turns out that this is quite pessimistic.

3 Deriving the Euler-Maclaurin Summation Formula

The Euler-Maclaurin Summation Formula tells us **exactly** what the error above is. Recall that earlier in the course, we integrated by parts to obtain:

$$\begin{aligned}
\int_0^1 f(t)\ dt &= \int_0^1 \frac{d}{dt}\ \left(t-\frac{1}{2}\right) f(t)\ dt \\
&= \left(t-\frac{1}{2}\right) f(t)|_0^1 - \int_0^1 \left[\frac{1}{2}\left(t-\frac{1}{2}\right)^2\right]' f'(t)\ dt \\
&= \left(t-\frac{1}{2}\right) f(t)|_0^1 - \frac{1}{2}\left(t-\frac{1}{2}\right)^2 f'(t)|_0^1 + \int_0^1 \left[\frac{1}{3!}\left(t-\frac{1}{2}\right)^3\right]' f''(t)\ dt \\
&= \left(t-\frac{1}{2}\right) f(t) - \frac{1}{2}\left(t-\frac{1}{2}\right)^2 f'(t) + \frac{1}{3!}\left(t-\frac{1}{2}\right)^3 f''(t) - \frac{1}{4!}\left(t-\frac{1}{2}\right)^4 f'''(t) + \cdots |_0^1
\end{aligned}$$

But the factors in parentheses are $\pm\left(\frac{1}{2}\right)^k$ at $t = 0, 1$, acting as averages and differences at the two points on the interval. Numerically evaluating, this gives us:

$$\begin{aligned}
\int_0^1 f(t)\ dt = &\frac{1}{1!}\left(\frac{1}{2}\right)^1 [f(1)+f(0)] \\
&- \frac{1}{2!}\left(\frac{1}{2}\right)^2 [f'(1)-f'(0)] \\
&+ \frac{1}{3!}\left(\frac{1}{2}\right)^3 [f''(1)-f''(0)] \\
&- \frac{1}{4!}\left(\frac{1}{2}\right)^4 [f'''(1)-f'''(0)] + \cdots
\end{aligned}$$

Thus we conclude that the Trapezoidal Rule gives us:

$$\frac{1}{2}[f(1)-f(0)] = \int_0^1 f(t)\ dt + \frac{1}{2!}\left(\frac{1}{2}\right)^2 [f'(1)-f'(0)] - \frac{1}{3!}\left(\frac{1}{2}\right)^3 [f''(1)+f''(0)] - \cdots$$

We get a telescoping (cancelling) series:

$$\begin{aligned}\left[\sum_{j=0}^{n} f(j)\right] &= \frac{1}{2}\left(f(1)-f(0)\right)+\frac{1}{2}\left(f(2)+f(1)\right)+\cdots+\frac{1}{2}\left(f(n)+f(n-1)\right)\\ &= \int_0^n f(x)\ dx+\frac{1}{2!}\left(\frac{1}{2}\right)^2\left[f'(n)-f'(0)\right]\\ &\quad -\frac{1}{3!}\left(\frac{1}{2}\right)^3\sum_{j=0}^{n} f''(j)\\ &\quad +\frac{1}{4!}\left(\frac{1}{2}\right)^4\left[f'''(n)-f'''(0)\right]\\ &\quad -\frac{1}{5!}\left(\frac{1}{2}\right)^5\sum_{j=0}^{n} f'''(j)+\cdots\end{aligned}$$

We have the actual integral plus some error, where half the error terms are beautiful, and don't look beautiful. So we want some way to 'kill' the even-numbered derivatives.

$$\frac{1}{2}\left[f(1)-f(0)\right]=\int_0^1 f(t)\ dt+\frac{1}{2!}\left(\frac{1}{2}\right)^2\underbrace{\left[f'(1)-f'(0)\right]}_{\text{good}}-\frac{1}{3!}\left(\frac{1}{2}\right)^3\underbrace{\left[f''(1)+f''(0)\right]}_{\text{bad}}-\cdots$$

$$\frac{1}{2}\underbrace{\left[f''(1)-f''(0)\right]}_{\text{bad}}=\int_0^1 f''(t)\ dt+\frac{1}{2!}\left(\frac{1}{2}\right)^2\left[f'''(1)-f'''(0)\right]-\frac{1}{3!}\left(\frac{1}{2}\right)^3\left[f''''(1)+f''''(0)\right]-\cdots$$

But notice

$$\sum_0^1 f''(t)\ dt=f'(1)-f'(0).$$

So we have:

$$\begin{aligned}\int_0^1 f(t)\ dt &= \frac{1}{2}\left[f(1)+f(0)\right]\\ &\quad +b_1\left[f'(1)-f'(0)\right]\\ &\quad +b_2\left[f^{(3)}(1)-f^{(3)}(0)\right]\\ &\quad +b_3\left[f^{(5)}(1)-f^{(5)}(0)\right]\\ &\quad +\cdots\end{aligned}$$

And we conclude the **Euler-Maclaurin Summation Formula**:

$$\int_0^n f(x)\ dx=\sum_{j=0}^{n} f(j)+b_1\left[f'(n)-f'(0)\right]+b_2\left[f^{(3)}(n)-f^{(3)}(0)\right]+\cdots$$

Remark: The idea is that no matter how long our integration interval is, the **error** from the Trapezoidal Rule only depends on the **endpoints**.

To kill the 'bad' averaging terms, because f is an arbitrary function, we 'plug in' f'' (double-prime), and we saw that we pushed the bad terms back by two.

Remark: Intuitively, because our error only depends on the endpoints, we can change the weights at the ends to be less (like $\frac{3h}{8}, \frac{5h}{8}, h, \ldots, h\frac{5h}{8}, \frac{3h}{8}$). More about this via 'Endpoint Correction'.

Lecture ends here.

We can see this neatly in Strain's handout on ECTR (Endpoint Corrections, Trapezoidal Rule).

Euler-Maclaurin formula Integrate by parts as in Taylor expansion:

$$\begin{aligned}
&\int_0^1 f(x)dx = \int_0^1 \frac{d}{dx}(x-1/2)f(x)dx \\
= \;& (1/2)(f(0)+f(1)) - \int_0^1 (x-1/2)f'(x)dx \\
= \;& (1/2)(f(0)+f(1)) - \int_0^1 \frac{d}{dx}\frac{1}{2}(x-1/2)^2 f'(x)dx \\
= \;& (1/2)(f(0)+f(1)) - \frac{1}{2}(1/2)^2(f'(1)-f'(0)) + \int_0^1 \frac{d}{dx}\frac{1}{3!}(x-1/2)^3 f''(x)dx \\
= \;& (1/2)(f(0)+f(1)) - \frac{1}{2!}(1/2)^2(f'(1)-f'(0)) + \frac{1}{3!}(1/2)^3(f''(1)+f''(0)) - \int_0^1 \frac{d}{dx}\frac{1}{4!}(x-1/2)^4 f'''(x)dx
\end{aligned}$$

and so forth. Rearrange to get an error formula for the trapezoidal rule:

$$(1/2)(f(0)+f(1)) = \int_0^1 f(x)dx + \frac{1}{2!}(1/2)^2(f'(1)-f'(0)) - \frac{1}{3!}(1/2)^3(f''(1)+f''(0)) + \int_0^1 \frac{d}{dx}\frac{1}{4!}(x-1/2)^4 f'''(x)dx.$$

Apply the formula to f'' in place of f:

$$\begin{aligned}
&(1/2)(f''(0)+f''(1)) = \int_0^1 f''(x)dx + \frac{1}{2!}(1/2)^2(f'''(1)-f'''(0)) - \frac{1}{3!}(1/2)^3(f''''(1)+f''''(0)) + \cdots \\
= \;& f'(1) - f'(0) + \frac{1}{2!}(1/2)^2(f'''(1)-f'''(0)) - \frac{1}{3!}(1/2)^3(f''''(1)+f''''(0)) + \int_0^1 \frac{d}{dx}\frac{1}{4!}(x-1/2)^4 f'''''(x)dx.
\end{aligned}$$

Key step: Use the result to eliminate the term involving $f''(1)+f''(0)$ from the previous formula:

$$(1/2)(f(0)+f(1)) = \int_0^1 f(x)dx + (1/12)(f'(1)-f'(0)) + \frac{1}{2!}(1/2)^2(f'''(1)-f'''(0)) + \int_0^1 \frac{d}{dx}\frac{1}{4!}(x-1/2)^4 f'''(x)dx.$$

Now imagine repeating the elimination infinitely often. The result would be to eliminate all the terms with plus signs between even derivatives of f and leave an infinite series of the form

$$(1/2)(f(0)+f(1)) = \int_0^1 f(x)dx + b_1(f'(1)-f'(0)) + b_2(f'''(1)-f'''(0)) + b_3(f'''''(1)-f'''''(0)) + \cdots$$

with some unknown constants $b_1 = 1/12$, b_2, b_3, ..., multiplying differences of odd-numbered derivatives of f. The Euler-Maclaurin summation formula follows by compounding:

$$\frac{1}{2}f(0) + f(1) + f(2) + \cdots + f(n-1) + \frac{1}{2}f(n) = \int_0^n f(x)dx + b_1(f'(n)-f'(0)) + b_2(f'''(n)-f'''(0)) + \cdots$$

because the differences of derivatives all telescope, canceling the interior terms. Conclusion: The error in the trapezoidal rule depends only on the derivatives of the integrand at the endpoints of the domain of integration. For example, the trapezoidal rule integrates a smooth periodic function over a full period with great accuracy.

ECTR It follows that the order of accuracy (degree of precision) of the trapezoidal rule can be increased by *endpoint corrections* which change the weights only near the endpoints of the interval. Such corrections can be derived by coupling the Euler-Maclaurin formula with finite difference approximations to the derivatives, or by polynomial interpolation as follows.

Let's use cubic interpolation to derive a fourth-order endpoint corrected trapezoidal rule. To do this, we interpolate four successive function values f_0, f_1, f_2, f_3 to integrate over the interval $[1, 2]$. Since the Lagrange basis functions are

$$L_0(x) = (x-1)(x-2)(x-3)/(0-1)(0-2)(0-3) = (x-1)(x-2)(x-3)/(-6)$$

$$L_0(x) = (x-1)(x-2)(x-3)/(0-1)(0-2)(0-3) = (x-1)(x-2)(x-3)/(-6)$$

$$L_1(x) = (x-0)(x-2)(x-3)/(1-0)(1-2)(1-3) = (x-0)(x-2)(x-3)/2$$
$$L_2(x) = (x-0)(x-1)(x-3)/(2-0)(2-1)(2-3) = (x-0)(x-1)(x-3)/(-2)$$
$$L_3(x) = (x-0)(x-1)(x-2)/(3-0)(3-1)(3-2) = (x-0)(x-1)(x-2)/6$$

the resulting rule is

$$\int_1^2 f(x)dx = w_0 f(0) + w_1 f(1) + w_2 f(2) + w_3 f(3)$$

where

$$w_0 = \int_1^2 L_0(x)dx = -1/24 = w_3$$

and

$$w_1 = \int_1^2 L_1(x)dx = 13/24 = w_2.$$

At the end intervals such as $[0,1]$ we do not have $f(-1)$ so we drop to quadratic interpolation with

$$L_0(x) = (x-1)(x-2)/(0-1)(0-2) = (x-1)(x-2)/2$$

$$L_1(x) = (x-0)(x-2)/(1-0)(1-2) = (x-0)(x-2)/(-1)$$
$$L_2(x) = (x-0)(x-1)/(2-0)(2-1) = (x-0)(x-1)/2$$

The resulting rule is

$$\int_0^1 f(x)dx = w_0 f(0) + w_1 f(1) + w_2 f(2)$$

where

$$w_0 = 10/24, \qquad w_1 = 16/24, \qquad w_2 = -2/24.$$

Putting it all together gives the fourth-order endpoint corrected trapezoidal rule

$$\int_0^1 f(x)dx = \frac{h}{24}\left(9f(0) + 23f(h) + 28f(2h) + 24f(3h) + 24f(4h) + \cdots + 9f(nh)\right).$$

Math 128A, Summer 2019
Lecture 14, Tuesday 7/16/2019

1 Review

Half-hour spent on talking over nilpotent matrices and linear algebra (Jordan Canonical Form), tangential from the course material.

2 Gaussian Integration

Yesterday we talked about Euler-Maclaurin. One of the particularly useful applications of the Euler-Maclaurin integration formula is to estimate discrete infinite sums.
Today we'll talk about Gaussian Integration, which we use to optimize numerical integration rules. Consider

$$\int_{-1}^{1} f(t)\ dt = \sum_{j=0}^{n} w_j f(t_j) + \underbrace{c f^{(2n+2)}(\xi)}_{\text{error}}$$

Normally we choose the weights w_j to get high accuracy on equispaced point. In Gaussian integration, we also increase the accuracy via choosing where to place points t_j.
This is exact for:

$$f(t) = 1; \sum w_j = 2$$
$$f(t) = t; \sum w_j t_j = 0$$
$$f(t) = t^2; \sum w_j t_j^2 = \frac{2}{3}$$

The idea here is to choose $(n+1)$ weights w_j and $(n+1)$ points t_j. This gives a horribly non-linear system of equations. We ought to be able to solve this for up to $(2n+2)$ non-linear equations and $(2n+2)$ variables $t_0, \ldots, t_n$. That means this will be exact for polynomials of degree $\leq 2n+1$. We know this because testing on $f(t) = t^{2n+1}$,

Remark: If we scale down to intervals of length h, we have sums of this form:

$$\int_{-h}^{h} f(x)\ dx = h \sum_{j=0}^{n} w_j f(t_j h) + c h^{2n+3} f^{(2n+2)}(\xi)$$

We just solved for the 'primal' example of Gaussian Integration. We chose a symmetric interval $[-1, 1]$ just so that by parity, half of our results equal to 0.
Now we try Hermite Interpolation to optimize the numertcal integration rule.

We'll build some polynomial $p(t)$ with:

$$p(t) = \sum_{j=0}^{n} A_j(t) f_j + B_j(t) f_j'$$

Recall that Hermite interpolation takes the function and its derivative and matches those values at our interpolation points (which we haven't decided yet).
Last time (Lec 11), we found:

$$A_j(t) = \left(1 - 2L_j'(t_j)(t - t_j)\right)^2 L_j(t)$$
$$B_j(t) = (t - t_j) L_j(t)^2$$

Thus:

$$\begin{aligned}\int_{-1}^{1} p(t)\ dt &= \sum_{j=0}^{n} \int_{-1}^{1} A_j(t)\ dt f_j + \int_{-1}^{1} B_j(t)\ dt f_j' \\ &= \sum_{j=0}^{n} \omega_j f_j + \underbrace{b_j}_{0} f_j'\end{aligned}$$

Gauss's idea is to set $b_j = 0$. That is,

$$b_j = \int_{-1}^{1} B_j(t)\ dt = \lambda_j \underbrace{\int_{-1}^{1} \omega(t) L_j(t) = 0}_{\forall 0 \le j \le n}$$

we have this property for all j if and only if for degree $p \le n$:

$$\iff \int_{-1}^{1} \underbrace{p(t)}_{\sum_{j=0}^{n} p(t_j) L_j(t)}\ dt = 0$$

This property above is called 'orthogonality' (inner product). That is, if we take:

$$x^T y = x_1 y_1 + \cdots + x_n y_n = 0,$$

then we say x is perpendicular to y (orthogonal). We like orthogonal functions because this is essentially linear independence 'to the max'; that is, projections to each other net 0.
We choose $t_0, \ldots, t_n$ such that

$$\int_{-1}^{1} \omega(t) p(t)\ dt = 0$$

That is, our $\omega(t)$ we construct is orthogonal to the degree $p \le n$ polynomial. Once we achieve this goal, we'll achieve the Gaussian integration rule, because the weights $w_0, \ldots, w_n$ are given by the exact same formula as A:

$$\begin{aligned} w_j &= \int_{-1}^{1} A(t)\ dt \\ &= \int_{-1}^{1} \left(1 - 2L_j'(t_j) \underbrace{(t - t_j)}\right) \underbrace{L_j(t)^2}\ dt \\ &= \int_{-1}^{1} L_j(t)^2\ dt > 0 \end{aligned}$$

We can also ignore the power 2 to get the weights, but we leave this here because this guarantees nonnegativity. Getting to the last line, we ignore the $-2L_j'(t_j)$ because that will not contribute to $B_j(t) = 0$.

Step 1: Find a representation of $\omega(t)$ in our favorite basis.
Step 2: Then our desired $t_0, \dots, t_n$ are simply the roots (zeros) to $\omega(t)$.

Newton's method does not work well. The standard monomial basis is never good. The Chebyshev basis could be good, but we might not know what to do (in order to achieve our goal).

Gauss suggests that we use ω for fewer points as a basis for degree n polynomials.

For example, consider the simple example of $n = 0$.

$$\begin{aligned}
\omega(t) &= t - t_0 =: P_1(t) \\
&\int_{-1}^{1} (t - t_0)p(t)\ dt = 0, \quad \deg p \le 0 \\
&\implies t_0 = 0; \\
&w_0 = \int_{-1}^{1} 1^2\ dt = 2
\end{aligned}$$

This nets us

$$\begin{aligned}
\frac{1}{2}(t - t_0)^2 \Big|_{-1}^{1} &= 0 \\
(1 - t_0)^2 &= (-1 - t_0)^2 \\
1 - 2t_0 + t_0^2 &= 1 + 2t_0 + t_0^2 \\
t_0 &= 0.
\end{aligned}$$

We write that $P_j(t)$ above is the (monic) "Legendre polynomial" of degree j. And we define $P_0(t) := 1$ (zero-degree) polynomial.
We will construct the $P_j(t)$ for 1 point, 2 points, and onwards. Then $P_{j+1}(t)$ will be easy just using orthogonality.
Now solve:

$$\int_{-1}^{1} \underbrace{P_{n+1}(t)}_{\omega(t) := 1 \cdot t^{n+1} + \cdots;\deg\ n+1; n+1\ \text{pts}} P_j(t)\ dt = 0; \quad 0 \le j \le n$$

We try

$$\overbrace{P_{n+1}(t) = \underbrace{[t]}\ P_n(t) + c_n P_{n-1}(t)}^{\text{will be our recurrence relation}} + d_n P_{n-2}(t) + \cdots$$

$$= t^{n+1} + \cdots$$

We need to boost the degree up by 1 somehow, so we insert this t. Each of these terms is orthogonal to each of the lower-degree polynomials. That is, all $j < k$, P_j, P_k are orthogonal. The coefficient for $P_n(t)$ has no numeric factor because we built this as a monic polynomial.
Then, we have:

$$\int_{-1}^{1} P_{n-1}(t)P_j(t)\ dt = 0; \quad j < n - 1$$

and

$$\int_{-1}^{1} \left[tP_n(t) + c_nP_{n-1}(t) + d_nP_{n-2}(t) + \cdots\right] P_j = 0; \qquad 0 \le j \le n$$

And for all $j < n-1$, by orthogonality and linearity, we have:

$$\int_{-1}^{1} P_n(t)tP_j(t)\ dt = 0.$$

The only interesting combination (multiplication) is:

$$\int_{-1}^{1} \left[tP_n + c_nP_{n-1}\right] P_{n-1} = 0; \qquad j = n-1$$

Consider

$$\int_{-1}^{1} tP_nP_n = 0 \implies \int_{-1}^{1} tP_n^2(t) = 0.$$

This is because $P_n(t)$ has even degree, and thus $tP_n^2(t)$ has odd degree, hence the symmetric integration equals 0. Then this suggests to us a parity between evens and odds, which brings us back to :

$$\begin{aligned} P_{n+1}(t) &= \underbrace{[t]}\ P_n(t) + c_nP_{n-1}(t) + \overbrace{d_nP_{n-2}(t) + \cdots}^{0} \\ &= t^{n+1} + \cdots \end{aligned}$$

and we know that $d_n = 0$ and onwards are all zero. To see this explicitly, see:

$$\begin{aligned} \int_{-1}^{1} \left[tP_n(t) + c_nP_{n-1}(t) + d_nP_{n-2}(t)\right] P_{n-2}(t)\ dt &= 0 \\ \int_{-1}^{1} P_n(t) \left[\underbrace{tP_{n-2}(t)}_{\deg\ n-1}\right]\ dt &= 0 \\ \int_{-1}^{1} c_nP_{n-1}(t)P_{n-2}(t)\ dt &= 0 \\ \int_{-1}^{1} d_nP_{n-2}^2(t)\ dt &= 0 \\ \implies d_n = 0 \implies e_n = 0 \implies f_n = 0 \implies \cdots & \end{aligned}$$

We get this result particularly from our strong induction construction of Legendre polynomials on orthogonality.
Consider, for $P_0(t) := 1, P_1(t) := t$, the

Recurrence relation:

$$P_{n+1}(t) = tP_n(t) - c_nP_{n-1}(t)$$

This recurrence relation

$$\int_{-1}^{1} tP_n(t)P_{n-1}(t)\ dt = c_n \int_{-1}^{1} P_{n-1}^2(t)\ dt$$

$$\implies c_n := \frac{\int_{-1}^{1} tP_n(t)P_{n-1}(t)\ dt}{\int_{-1}^{1} P_{n-1}^2(t)\ dt}$$

So

$$c_1 = \frac{\int_{-1}^{1} tP_1(t)P_0(t)}{\int_{-1}^{1} P_0^2} = \frac{2/3}{2} = \frac{1}{3}$$

Hence $P_2(t) = t^2 - \frac{1}{3}$.
And thus our **Two-Point Gauss Rule** gives us:

$$t_0 = \frac{1}{\sqrt{3}}; \qquad w_0 = 1$$
$$t_1 = \frac{-1}{\sqrt{3}}; \qquad w_1 = 1.$$

It turns out, with Legendre polynomials:

$$c_n = \frac{n^2}{4n^2 - 1}$$

Then testing this, we see $c_2 = \frac{4}{15}$, and

$$\begin{aligned} P_3(t) &= tP_2(t) - \frac{4}{15}P_1(t) \\ &= t^3 - \frac{1}{3}t - \frac{4}{15}(t) = t^3 - \frac{3}{5}t \\ &= t\left(t - \sqrt{\frac{3}{5}}\right)\left(t + \sqrt{\frac{3}{5}}\right) \end{aligned}$$

And if we want a 3-point Gaussian quadrature, we can use

$$t_0 := 0 \qquad t_1 := \sqrt{\frac{3}{5}} \qquad t_2 := -\sqrt{\frac{3}{5}}$$

We would expect: $w_1 = w_2$. Then

$$\begin{aligned} w_0 &= \int_{-1}^{1} L_0(t)\ dt \\ &= \int_{-1}^{1} \frac{(t - t_1)(t - t_2)}{(t_0 - t_1)(t_0 - t_2)}\ dt \\ &= \frac{-5}{3}\int_{-1}^{1} t^2 - \underbrace{t_1 + t_2}_{0} + t_1 t_2\ dt \\ &= -\frac{5}{3}\left(\frac{2}{3} - \frac{3}{5}2\right) = \frac{8}{9} \end{aligned}$$

We have $w_0 + w_1 + w_2 = 2$ (from earlier), so $w_1 = w_2 = \frac{5}{9}$.
Thus we say that we built the inductive proof of a theorem:

Theorem 2.1. For any $n \geq 0$, there are points $t_0, \ldots, t_n \in [-1, 1]$ and weights $w_0, \ldots, w_n > 0$ with:

$$\int_{-1}^{1} f(t)\ dt = \sum_{j=0}^{n} w_j f(t_j),$$

when the degree of $f \leq (2n + 1)$.

Recall that we can only find roots explicitly for degree up to 5, but we have one at zero, so secretly this becomes just a quadratic; two positive, two negative.
It turns out, for us we can take up to 9; past this, for higher degree polynomials, we solve numerically.

Midterm topics: fixed point iteration, big-O, floating point; interpolation ; no questions on numerical integration ; dont need integral IVT.
New problems! Not just old mt problems, but re-using 1.

Math 128A, Summer 2019

Lecture 15, Wednesday 7/17/2019

CLASS ANNOUNCEMENTS: Midterm in-class today; No lecture.

Math 128A, Summer 2019
Lecture 16, Thursday 7/18/2019

Topics Today:

- ECTR
- Gaussian Integration
- Adaptive Integration

1 Midterm Problem 2

This is a model problem for when we want to look at the distances between things like us, satellites and space stations. Strain gives a beautiful explanation that I (in retrospect should have but) did not type up.
The summation

$$\begin{aligned} g_i &= \sum_{j=1}^{n} \frac{f_i}{x_i - y_j}; \qquad O(n^2) \\ &= \sum_{j=1}^{n} \frac{1}{x_i} \sum_{q=0}^{p-1} \underbrace{\left(\frac{y_j}{x_i}\right)^q}_{y_j^{\varepsilon}\left(\frac{1}{x_i}\right)^{\varepsilon}} + O(\varepsilon) \end{aligned}$$

has been proven that we need order of $O(n^2)$ work. For precision to the order of $\varepsilon = 2^{-52}$, it turns out we need $p \geq 52$.
The first rule of applied mathematics is to swap order of sums to see what this gives.
Notice because y is small and x is tiny, we write:

$$\frac{1}{x-y} = \frac{1}{x}\frac{1}{1-\frac{y}{x}} = \frac{1}{x}\sum_{q=0}^{\infty}\left(\frac{y}{x}\right)^p.$$

2 Endpoint Corrected Trapezoidal Rule (ECTR)

The first part of the homework looks at the Euler-Maclaurin summation formula.

$$\sum_{0}^{1} f(t)\ dt = \frac{1}{2}\left[f(0) + f(1)\right] + b_1\left[f'(1) - f'(0)\right] + b_2\left[f'''(1) - f'''(0)\right]$$

When we compound these, we get:

$$\int_0^n f(x)\ dx = \underbrace{\left[\sum_{0}^{n} f(j)\right]''} + b_1\left[f'(n) - f'(0)\right] + b_2\left[f'''(n) - f'''(0)\right] + \cdots$$

$$\sum_{j=0}^{n} f(j) = \frac{1}{2}f(0) + f(1) + \cdots + f(n-1) + \frac{1}{2}f(n)$$

We plug into $f(x)\ dx = e^{-tx}\ dx$, using by-parts: $\frac{d}{dx}e^{-tx}$.

$$\begin{aligned}\int_0^\infty e^{-tx}\ dx = \frac{1}{t} &= \left[\sum_{j=0}^{\infty}\right]'' + b_1\left[-f'(0)\right] + b_2\left[-f'''(0)\right] + \cdots \\ &= \frac{-1}{2} + \frac{1}{1-e^{-t}} + b_1 t + b_2 t^3 + \cdots\end{aligned}$$

Essentially, we are plugging in the function e^{-tx} to 'extract the b's ', because we know if it works for everything, then it must work for our example. Multiplying out by t, we have:

$$t\sum_0^\infty e^{tx}\ dx + \frac{t}{2} = \frac{t}{1-e^{-t}}$$

Then we set the bernoulli numbers b_i:

$$b_1 t^2 + b_2 t^4 + b_3 t^6 + \cdots = 1 + \frac{t}{2} - \frac{t}{1-e^{-t}}$$

The explicit formula for the bernoulli numbers is:

$$\begin{aligned} b_1 &= \frac{B''(0)}{2!} \\ b_2 &= \frac{B''''(0)}{4!} \\ &\vdots\end{aligned}$$

but we don't want to do this kind of math, even if it's finite, because taking the derivatives can be tedious. We do this another way:

$$\int_0^1 f(t)\ dt = \int_0^{nh} f(t)\ dt = h\left[\sum_{j=0}^{n} f(jh)\right]'' + \underbrace{b_1 h^2\left[f'(nh) - f'(0)\right]} + b_2 h^4\left[f'''(nh) - f'''(0)\right] + \cdots$$

The mean value theorem gives us that the underbraced expression is equal to $-\frac{1}{12}h^2 f''(\xi)$, so we know $b_1 = \pm\frac{1}{12}$ (one of these, Strain can't remember). We then toss in everything we know about the error:

$$\int_0^1 f(t)\ dt = h\left[\sum f(jh)\right]'' - \underbrace{\frac{1}{12}h^2\left[f'(nh) - f'(0h)\right]} + O(h^4).$$

We approximate this with the cheapest possible approximation (1 point):

$$f'(nh) = \frac{f(nh) - f([n-1]h)}{h} + O(h^1),$$

whereas if we use two points to make a linear approximation. However, this order of accuracy is not enough, because multiplying into our above expression does not match $O(h^4)$.

Consider

$$f_0' = af_0 + bf_1 + cf_2.$$

A cheap and dirty way to find these coefficients, we try: $f(t) = 1$, $f(t) = t$, and $f(t) = t^2$. If we're lucky, we get three equations and three unknowns.

$$\begin{aligned} 0 &= a + b + c \\ 1 &= a \cdot 0 + b \cdot 1 + c \cdot 2 \\ 0 &= a \cdot 0^2 + b \cdot 1^2 + c \cdot 2^2 \end{aligned}$$

This gives a system of linear equations, netting:

$$a = \frac{-3}{2}, \qquad b = 2, \qquad c = \frac{-1}{2}.$$

This is sufficient to get a fourth-order integration rule. Bringing this back into the main question, we have:

$$\begin{aligned} &h\left[\frac{1}{2}f_0 + f_1 + f_2 + \cdots + \frac{1}{12}\left(\frac{-3}{2}\right)f_0 + \frac{1}{12}(2)f_1 + \frac{1}{12}\left(\frac{-1}{12}\right)f_2\right] \\ &= \frac{h}{24}[9f_0 + 28f_1 + 23f_2 + 24f_3 + 24f_4 + 24f_5 + \cdots \\ &\qquad \cdots + 24f_{n-4} + 24f_{n-3} + 23f_{n-2} + 28f_{n-1} + 9f_n] + O(h^4). \end{aligned}$$

We corrected 3 weights at each end, symmetrically. We only need to do the calculation at one end and read off backwards for the other end.
Compounding is how we get the order of h. By endpoint correcting, we get a much better result than we do in Simpson's rule, which uses quadratic interpolation.

Remark: Strain doesn't actually like ECTR, but it's relatively good when we need to use equispaced points. Using equal weights (except at the endpoints) is the most 'stable' thing we can do, when we do things like fast fourier transforms.

3 Gaussian Integration

As a quick review, recall that our 'recipe' was really simple. Let

$$P_0 := 1, \qquad P_1 := t; \qquad P_{n+1}(t) = t \cdot P_n(t) - c_n$$

Let our points $t_0, \ldots, t_n$ be the roots of the polynomial P_{n+1}, and our weights

$$\begin{aligned} w_j &:= \int_{-1}^{1} L_j(t)\ dt \qquad \text{by hand; } n \leq 4 \\ &= \int_{-1}^{1} [L_j(t)]^2\ dt \end{aligned}$$

Remark: As an aside, when performing bisection, we notice that for polynomials, the roots of some polynomial P_n are 'trapped' between the roots of polynomial P_{n+1}. We say that they are '**interlaced**'.

This result leads us to know that for $t_0^n, \ldots, t_{n-1}^n$ roots of P_n,

$$-1 < t_0^{n+1} < t_0^n < t_1^{n+1} < t_1^n < \cdots < t_{n-1}^n < t_n^{n+1} < 1$$

We take the exact P_n as given by Gaussian Quadrature. The reason this 'interlacing' occurs is due to the fact that P_k is orthogonal to all P_r for all $r < k$. Orthogonal functions are tightly related to integration points, within an interval, for example $[-1, 1]$. If we want to prove the interlacing phenomenon, we can take the differential equations of the Legendre polynomial.
Strain's favorite expression for $w_j = \int_{-1}^{1} L_j(t)\ dt$ is:

$$\begin{aligned} w_j &= \int_{-1}^{1} \lambda_j \frac{\omega(t)}{t - t_j}\ dt \\ &= \lambda_j \int_{-1}^{1} \frac{P_{n+1}(t)}{t - t_j}\ dt \end{aligned}$$

Recall that we defined $\omega(t) =: P_{n+1}(t)$ in class. It turns out that this is an interesting formula. We may not want to use this for numerical analysis (and this class), as we are dividing by $t - t_j$.

4 Adaptive (Gaussian) Integration

The idea is very simple in that we want to capture all the 'wacky' behavior of functions on subintervals. We'll say:

$$\begin{aligned} \int_a^b f(x)\ dx &= \sum_{j=1}^{N} \int_{a_j}^{b_j} f(x)\ dx \\ &= \sum_{j=1}^{n} h_j \end{aligned}$$

Let:

$$\int_a^b f(x)\ dx = h \int_{-1}^{1} f(m + th)\ dt$$

$h_j := \frac{b_j - a_j}{2}$ stands for half-length of the interval. To check if our multiplication by h at the left is correct, we check a function that we know:

$$\int_a^b 1\ dx = b - a = h \int_{-1}^{1} 1\ dt = 2h$$

Hence we write:

$$\begin{aligned} \int_a^b f(x)\ dx &= h \int_{-1}^{1} f(m + th)\ dt \\ &= \underbrace{h_j \sum_{q=0}^{p} w_q f(m_j + t_q h_j)}_{\text{Gaussian Quadrature}} + O\left(h_j^{2p+1} f^{(2p)}(\xi_j)\right) \end{aligned}$$

To see again (differently) how we assigned/defined a, b above, consider the change-of-variables:

$$\begin{aligned} x &= m + th \\ dx &= h \cdot dt \\ t &= -1 \qquad x = m - h =: a \\ t &= 1 \qquad x = m + h =: b. \end{aligned}$$

Then this nets us

$$\int_a^b f(x)\ dx = h \sum_{-1}^{1} f(m + th)\ dt$$

So our main equation is:

$$\int_a^b f(x)\ dx = \sum_{j=1}^{n} h_j \sum_{q=0}^{p} w_q f(m_j + t_q h_j) + O\left(h_j^{2p+1} \underbrace{f^{(2p)}(\xi_j)}_{\text{we don't know}} \right)$$

We choose intervals $[a_j, b_j]$ to subdivide $[a, b]$, where

$$[a, b] = \bigcup_{j=1}^{n} [a_j, b_j]$$

An easier way to do this than manually setting intervals is to define a recursive subdivision of the interval, $I(a, b)$. To do this, we break an interval into:

$$[\ \underbrace{a_1}\qquad \underbrace{b_1 = a_2 = m}\qquad \underbrace{b_2}\ ]$$

Doing so, we get an error estimate of:

$$\text{error} = I(a, b) - [I(a_1, b_1) + I(a_2, b_2)]\,.$$

and we define that if the absolute value of our error estimate is less than a user-specified tolerance, then we're done. If not, then we keep on going (we declare this was progress and keep on subdividing).
One way to do this (computer science) is to build a **stack**; that is, a *list* of intervals:

$$L = (\{[a_1, b_1], I_1\}, \{[a_2, b_2], I_2\}, \ldots)$$

Our step is to take the next element of the stack if the error estimate is less than the tolerance. Else, split into two new elements; that is, stack new elements at end and move on.
The class went on to talk about heaps, stacks, queues, and the differences thereof.

Remark: Recall that when we built `bisection.m`, we accepted (but did not use) a parameter p. We'll be using a parameter of Legandre polynomials (and we'll need to specify which one) to pass into `bisection.m`.

Lecture ends here.

Math 128A, Summer 2019
Lecture 17, Monday 7/22/2019

CLASS ANNOUNCEMENTS: Topics today:
- Review
- Weights in numerical integration (quadrature)

1 Review

From the previous homework, recall the question on Differential coefficients.

$$\delta_{n,j}^m = \cdots \delta_{n-1,j}^m + \cdots \delta_{n-1,j}^{m-1}$$

This gives:

$$f^{(m)}(a) = \sum_{j=0}^{N} \delta_{N,j}^m f(t_j).$$

We start from one 'corner' and build successive columns (or triangles), successively larger and larger by 1.

1.1 Homework 5

Let's say we want to calculate the Taylor expansion of some horrible function, say

$$\frac{t}{e^t - 1} = \sum_{m=0}^{\infty} a_m t^m.$$

The way a human-being should do this is to cross-multiply and say:

$$\begin{aligned} t &= (e^t - 1) \sum_{m=0}^{\infty} a_m t^m \\ &= \left(\sum_{k=0}^{\infty} \frac{t^k}{(k+1)!} \right) \left(\sum_{m=0}^{\infty} a_m t^m \right) \\ &= \left(\frac{1}{1!} + \frac{t}{2!} + \frac{t^2}{3!} + \cdots \right) \left(a_0 + a_1 t + a_2 t^2 + \cdots \right) \end{aligned}$$

This gives an infinite system of linear equations for the a's, so this would be very scary, except that this is a **triangular** system of linear equations. That is, a_0 is determined right away, and a_1 is given by looking at the linear terms. Hence we can write this neatly as a recurrence relation (although it doesn't look like a normal recurrence relation). Each element in the solution depends on every previous element.
As applied to our homework, this is how we calculate our Bernoulli coefficients, $b_0, \ldots, b_{10}$. Else, for our homeworks, we can plug this into Mathematica and tell it to Taylor expand symbolically.
Without this 'trick' of multiplying by $(e^t - 1)$, we can taylor expand the entire $\frac{t}{e^t - 1}$. These Bernoulli numbers have been proven to not have 'nice'

recurrence relation forms but rather requires all previous terms (combinatorics).
In our previous example, this gives a relation something like:

$$\sum_{j=0}^{k} \frac{a_j}{(k+1-j)!} = 0.$$

The triangular system of equations arises like:

$$\begin{aligned} 1 &= \frac{a_0}{1!} \\ 0 &= \frac{a_1}{1!} + \frac{a_0}{2!} \\ 0 &= \frac{a_2}{1!} + \frac{a_1}{2!} + \frac{a_0}{3!} = 0 \\ &\vdots \end{aligned}$$

1.2 Question 1: a,b

To find the recursive formula for Euler-Maclaurin,

$$\int_0^n d(x)\ dx = \left[\sum_{j=0}^{n} f(j)\right] - \sum_{m=1}^{n} b_{2m} f^{(2m-1)}(0).$$

We take $f(x) := e^{-\lambda x}$ or $f(x) := x^p$ to get b_{2m} if we know the power sums. In our problem set, we want to show that, in terms of the Bernoulli numbers, $\sum_{n=0}^{N} n^p$ is equal to the polynomial $P_{n+1}(n)$.

1.3 Problem 5

Recall the famous **monic Legendre polynomials**:

$$\begin{aligned} P_0(t) &:= 1 \\ P_1(t) &:= t \\ P_{n+1}(t) &:= tP_n(t) - c_n P_{n-1}(t), \qquad c_n := \frac{n^2}{4n^2-1}. \end{aligned}$$

Then $P_2(t) = t^2 - \frac{1}{3} \cdot 1 = t^2 - \frac{1}{3}$, and $P_3(t) = t^3 - \frac{3}{5}t$. Then later,

$$P_5(t) = t^5 + \cdots t^3 + \cdots t.$$

The idea for this problem is that by Galois, we know we cannot explicitly find zeroes for this quintic polynomial; however, our function is odd, so factoring out a t, then we have:

$$P_5(t) = tP_2(t^2),$$

and we know the formula for the quadratic polynomial. Hence we can find explicit forms of our roots.

2 Integration Weights

Strain shares with us here what he's been toying with over the weekend.
Let $t_0, \dots, t_n$ be the zeroes of $P_{n+1}(t) = t^n + \cdots$, and $L_j(t_k) = \delta_{j,k}$, where

$$L_j(t) = \frac{\prod_{k \neq j}(t - t_k)}{\prod_{k \neq j}(t_j - t_k)} = \lambda_j \frac{P_{n+1}(t)}{t - t_j},$$

and

$$P_n(t) = (t - t_0)(t - t_1) \cdots (t - t_n).$$

By chain rule, each time we differentiate, we differentiate one factor and leave the the rest, and we sum all of these results. Hence we write:

$$P_{n+1}(t) = \sum_{j=0}^{n} \prod_{k \neq j} (t_j - t_k).$$

Particularly,

$$\begin{aligned} P'_{n+1}(t_i) &= \prod_{k \neq i} (t_i - t_k) \\ &= \frac{1}{\lambda_i}. \end{aligned}$$

Then

$$L_j(t) = \frac{1}{P'_{n+1}(t_j)} \frac{P_{n+1}(t)}{t - t_j}.$$

Let

$$\begin{aligned} w_j &:= \int_{-1}^{1} L_j(t)\ dt \\ &= \frac{1}{P'_{n+1}(t_j)} \int_{-1}^{1} \frac{P_{n+1}(s)}{s - t_j}\ ds. \end{aligned}$$

If we can perform this integration (we may not want to do this via quadrature), then this integral is simply a function evaluation, $Q_{n+1}(t_j)$, where we write:

$$\begin{aligned} Q(t) &:= \int_{-1}^{1} \frac{P_{n+1}(s)}{s - t}\ ds - c_n Q_{n-1}(s) \\ &= \int_{-1}^{1} \frac{sP(s) - c_n P_{n-1}(s)}{s - t}\ ds. \end{aligned}$$

So it looks like a recurrence relation.
This is the integral-free formula for the integration weights of Gaussian Integration (Quadrature.)
So we write:

$$\begin{aligned} Q_{n+1}(t) &= \overbrace{\int_{-1}^{1} P_0(s) \cdot P_n(s)\ ds}^{0} + t \int_{-1}^{1} \frac{P_n(s)}{s - t}\ ds - c_n Q_{n-1}(t) \\ &= t_j Q_n(t_j) - c_n Q_{n-1}(t_j) =: w_j \end{aligned}$$

Then explicitly,

$$Q_0(t) = \int_{-1}^{1} \frac{1}{s-t}\, ds = \ln\left(\frac{1-t}{1+t}\right)$$
$$Q_1(t) = \int_{-1}^{1} \frac{s}{s-t}\, ds = 2 + Q_0(t).$$

Once we get these expressions,
While we are recurring to P_{n+1}, we can differentiate to get:

$$P_{n+1} = tP_n - c_n P_{n-1}$$
$$P'_{n+1} = tP'_n + P_n - c_n P'_{n-1}$$

Then

$$P'_0 = 0; \qquad P'_1 = 1.$$

5 Minute Break.

3 Ordinary Differential Equations (ODEs)

Let T be some (current) time.

$$\begin{aligned} y' &= f(t, y) \\ g &: [0, T] \to \mathbb{R} \\ f &: [0, T] \times \mathbb{R}^n \to \mathbb{R} \end{aligned}$$

Our y is a column vector:

$$y'(t) = \begin{bmatrix} y_1'(t) & \cdots & y_n'(t) \end{bmatrix}^T$$

Today we'll talk about the theory behind ODEs.

3.1 Three (3) Canonical Examples:

1. Linear, Homogeneous real function

$$y' = a(t)y + g(t)$$

This is linear in y, not necessarily linear in t. The solution always exists and is always unique, and we can simply write them down in terms of integrals, due to the the work of Euler-Maclaurin from last week.
One way to solve this equation is to remember the **integrating factor**: $e^{-A(t)}$, where $A'(t) := a(t)$.

$$\begin{aligned} e^{-A(t)} \left(y' - a(t)y\right) &= e^{A(t)} g(t) \\ \left(e^{-A(t)} y(t)\right)' &= e^{-A(t)} g(t), \end{aligned}$$

where we recognize the perfect derivative in the second line. We write:

$$\begin{aligned} \int_0^t \left(e^{-A(s)} y(s)\right)' \, ds &= e^{-A(t)} y(t) - e^{-A(0)} y(0) \\ &= \int_0^t e^{-A(s)} g(s) \, ds \end{aligned}$$

When we integrate 0 to t, do not call our variable of integration also t (here we use s). If we can't solve this analytically (exactly), then we know how to do it numerically (i.e. via Gaussian Integration).
So we have:

$$y(t) = e^{A(t) - A(0)} y_0 + \int_0^t e^{A(t) - A(s)} g(s) \, ds.$$

We conclude that **if a solution exists**, then it **must** satisfy this formula (provided our integrals above make sense). Hence this proves **existence** and **uniqueness** of our solution.
Existence and uniqueness is how we justify initial value problems like $y' = f(t, y)$ with $y(0) = y_0$.

(2) Cautionary Example of Nonexistence (of solution at a finite time).
Let

$$\begin{aligned} y' &= y^2 \\ y(0) &= y_0. \end{aligned}$$

We separate variables, to write: $\frac{y'}{y^2} = 1 = \text{independent}$.
Hence

$$\int_0^1 \frac{y'(s)\ ds}{[y(s)]^2} = \int_0^t 1\ ds,$$

so if y is monotone, then we can use it perfectly well as a variable of integration. That is, $dy = y'(s)ds$. So we write:

$$\int_{y_0}^{y(t)} \frac{du}{u^2} = \int_0^1 \frac{y'(s)\ ds}{[y(s)]^2} = \int_0^t 1\ ds = \frac{-1}{u}\Big|_{u=y(0)}^{u=y(t)},$$

Hence this gives us:

$$\begin{aligned} \frac{-1}{y(t)} + \frac{1}{y_0} &= t \\ \implies y(t) &= \frac{1}{\frac{1}{y_0} - t}. \end{aligned}$$

This tells us to watch out in the denominator, that this expression can blow up at $t = \frac{1}{y_0}$, if $y_0 > 0$.

(3) Non-uniqueness $\implies$ We Cannot Predict the Future in this Model

Let $y' := 2\sqrt{y}, \qquad y(0) = y_0 \geq 0$. We can proceed similarly as in (2). We can write:

$$\frac{y'}{2\sqrt{y}} = \sqrt{y}' = 1,$$

which gives us:

$$\sqrt{y(t)} - \sqrt{y_0} = t \implies y(t) = (t + \sqrt{y_0})^2$$

For example, $y_0 = 0$, so $y(t) = t^2$. But also, $y'(t) = 2t = 2\sqrt{y(t)}$ gives $y(t) = 0$.

We have two solutions, so which is correct?

Also consider:

$$y(t) = \begin{cases} (t-c)^2, & 0 < c \leq t \\ 0, & 0 \leq t \leq c. \end{cases},$$

where we have any $c \geq 0$. This is also a solution!!! Everything is zero, and $y' = 2\sqrt{y}$. We don't know if it stays 0 forever, or until an arbitrary time where it starts increasing.

Notice that is function is differentiable at c.

4 How to Rule Out Bad Behavior (Non-existence, Non-uniqueness)

Our answer is to try to prove existence and uniqueness and to see what assumptions we need to make about $f(t, y)$.
To prove existence and uniqueness, look at:

$$y'(t) := f(t, y), \qquad y(0) = y_0.$$

A useful thing to do here is to integrate this, to get:

$$\int_0^t y'(s)\ ds = \int_0^t f(s, y(s))\ ds,$$

and by FTC (fundamental theorem of calculus) on the LHS, we have:

$$y(t) = \underbrace{y_0 + \int_0^t f(s, y(s))\ ds}_{F}.$$

If we know y up to (and barely including t), then we have a solution that is equivalent to our given initial condition. We can see it works in both directions. We call this the Picard integral equation.
The single equation looks like a fixed-point equation. The idea is that we stick $y(s)$ into the integral and get the same thing. That is,

$$y = F[y]$$

Notice f takes on point values and gives a value. On the other hand, F takes a whole function and gives us back a whole function. So we can write something like:

$$y(t) = F[y](t),$$

as a fixed-point equation in the vector space of functions.
Explicitly, we define:

$$F : y \to y$$
$$F[y](t) := y_0 + \int_0^t f(s, y(s))\ ds$$

Before we try Newton's method, we are legally obligated to try fixed point iteration. We guess something and plug it into the RHS. So fixed point iteration would say:

$$y_{n+1}(t) = y_0 \int_0^t f(s, y_n(s))\ ds,$$

with $y_0(t) := y_0$.

Example: $y' = y^2, \qquad y(0) = 1$. Notice from earlier, we have this is exactly equivalent (in both directions) to

$$y(t) = 1 + \int_0^t y(s)^2\ ds$$

Then $y_0(t) = 1$, and

$$y_1(t) = 1 + \int_0^t y_0(s)^2 \, ds = 1 + \int_0^t 1 \, ds = 1 + t.$$

Hence

$$\begin{aligned} y_2(t) &= 1 + \int_0^t (1+s)^2 \, ds \\ &= 1 + \int_0^t [1 + 2s + s^2] \, ds \\ &= 1 + t + t^2 + \frac{t^3}{3} \end{aligned}$$

We guess: $y_3(t) = [1 + t + t^2 + t^3] + \cdots$, where we don't know what $\cdots$ is. But we can guess: $\to \frac{1}{1-t}$.

Lecture ends here.

Next time, we'll see how long we can guarantee good behavior, and how to do so.

Math 128A, Summer 2019
Lecture 18, Tuesday 7/23/2019

1 Review, ODEs

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ +y(0) &= y_0 \end{aligned}$$

Adding these gives an initial value problem:

$$\begin{aligned} y(t) &= y_0 + \int_0^t f(s, y(s))\ ds \\ &= F[y](t) \\ F[u](t) &= y_0 + \int_0^t f(s, u(s))\ ds \end{aligned}$$

2 Fixed Point Iteration (of Functions)

$$\begin{aligned} y_{n+1}(t) &= y_0 + \int_0^t f(s, y_n(s))\ ds \\ y_0(t) &= y_0 \end{aligned}$$

Convergent if : (1) Invariant set of y, (2) contractive. Suppose we are looking for a solution on the interval $0 \le t \le T$.
To See Contractive:

$$\sup_{0\le t\le T} |F[u] - F[v]| \le \frac{1}{2} \sup_{0\le s\le T} |u - v|.$$

To See Invariance If our function f is nice and bounded; that is, $|f(t, u)| \le M$, then:

$$|y_{n+1}(t) - y_0| \le Mt \le MT$$

Invariance is a lot harder to see in general, but luckily it is not as important to check, according to Strain. So when is F contractive?

$$\begin{aligned} F[u](t) - F[v](t) &= \int_0^t f(s, u(s))\ ds - y_0 - \int_0^t f(s, v(s))\ ds \\ &= \int_0^t [f(s, u(s)) - f(s, v(s))\ ]\, ds \end{aligned}$$

Contractive is to show that if two points in the domain are close, then their values at those points should be even closer. We want:

$$\left| \int_0^t f(s, u(s)) - f(s, v(s))\ ds \right| \le L|u(s) - v(s)|$$

Definition: Lipschitz -

We say a functin $f(t,u)$ is **Lipschitz** if and only if there is a constant L such that

$$|f(t,u) - f(t,v)| \leq L|u-v|,$$

for all u, v.

This is stronger than being continuous, but a tiny bit weaker than differentiable.
As an aside, if we have for $0 < \alpha < 1$,

$$|f(t,u) - f(t,v)| \leq L|u-v|^{\alpha},$$

then we call this **Hölder** α-continuity (pronounced 'Hurlder').
We see that this is the restriction we wanted to impose in our previous differential equations.

Theorem 2.1. If $|f(t,u)| \leq M$ and $|f(t,u) - f(t,v)| \leq L|u-v|$, then there is a **unique solution** to

$$y' = f(t,y) \qquad y(0) = y_0$$

on the interval $[0,\tau]$ where τ is a function $\tau(T,R,M,L) > 0$ on those variables, which can be found via dimensional analysis. These are $0 \leq t \leq T$, $u - y_0| \leq R$, and M, L as above.

3 Systems of ODEs and Higher-Order Equations

For example, $F = ma$ means:

$$y'' = \frac{1}{m}F(y, y')$$

If we stick to a single variable $y' = f(t, y)$, we are very much stuck, but if we are willing to go to a vector $y_i' = f_i(t, y_1, \ldots, y_n)$, where $i = 1 : n$, say

$$\begin{aligned} y'' &= \frac{1}{m}F(\underbrace{y}_{u}, \underbrace{y'}_{v}) \\ v' := y'' &= \frac{1}{m}F(u, v) \\ u' := y' &= v, \end{aligned}$$

this gives a system of equations that unstucks us:

$$\begin{bmatrix} u \\ v \end{bmatrix}' = \begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} v \\ \frac{1}{m}F(u, v) \end{bmatrix} = f(Y).$$

We don't need higher order differentiation coefficients or discretization.

Example: Last time we looked at

$$\begin{aligned} y' &= a(t)y + b(t) \\ y'' + \omega^2 y &= g(t), \end{aligned}$$

where we want to write the second equation (second order equation) as a first order problem.
Then we have, for $u := y, v := y'$,

$$\begin{aligned} u' &= v \\ v' &= -\omega^2 y + g(t) \\ &= -\omega^2 u + g(t). \end{aligned}$$

Hence:

$$\begin{bmatrix} u \\ v \end{bmatrix}' = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \begin{bmatrix} u & v \end{bmatrix} + \begin{bmatrix} 0 & g(t) \end{bmatrix}$$

gives $y' = Ay + g(t)$, where

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \\ A^2 &= \begin{bmatrix} -\omega^2 & 0 \\ 0 & -\omega^2 \end{bmatrix} = -\omega^2 I \end{aligned}$$

And further,

$$\begin{aligned} y' &= Ay + g(t) \\ y' - Ay &= g \\ \left(e^{-tA}y\right)' &= e^{-tA}g \\ e^{-tA}y(t) - e^{-0A}y(0) &= \int_0^t e^{-sA}g(s)\, ds \end{aligned}$$

This is almost a solution, but we don't have $y(t)$ isolated yet. That gives:

$$y(t) = e^{tA}y_0 + \int_0^t e^{(t-s)A}g(s)\ ds$$

This tells us how to deal with constant coefficient differential equations, with a forcing constant ω. We have many ways to express e^{tA}, but we noticed a nice pattern with A^2 earlier, so we take the exponential series (Taylor expansion):
Recall:

$$\begin{aligned} e^{\lambda} &= \sum_{n=0}^{\infty} \frac{\lambda^n}{n!} \\ &= \sum_{n=0}^{\infty} \frac{\lambda^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{\lambda^{2n+1}}{(2n+1)!}, \end{aligned}$$

where the left expression gives $\cosh\lambda$ and the right gives $\sinh\lambda$. Inserting i to get an oscillating term within each summation, we get:

$$\begin{aligned} e^{i\lambda} &= \sum_{n=0}^{\infty} \frac{\lambda^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{\lambda^{2n+1}}{(2n+1)!} \\ &= \cos\lambda + i\sin\lambda. \end{aligned}$$

So for our problem, we write (with ω under sin via dimensional analysis):

$$\begin{aligned} e^{tA} &= \cos(\omega t)I + \sin(\omega t)A \\ \frac{d}{dt}Ae^{tA} &= -\omega\sin(\omega t) + \sin(\omega t)A \\ Ae^{tA} &= \cos(\omega t)A + \sin(\omega t)A^2 \\ &= \cos(\omega t)A - \omega^2\sin(\omega t)I \\ &= \begin{bmatrix} -\omega^2\sin(\omega t) & \cos(\omega t) \\ -\omega^2\cos(\omega t) & -\omega^2\sin(\omega t) \end{bmatrix} \end{aligned}$$

We don't have an i preceding the sin terms because the eigenvalues of A^2 are already complex, in ω.
So, this gives:

$$\frac{d}{dt}e^{tA} = \underbrace{\omega}\, Ae^{tA}$$

However, we should not have ω here, so to fix this, we realize that our definitions for u, v had inconsistent dimensions with respect to ω. Instead, we should have:

$$u := y; \qquad v := \frac{1}{\omega}y'.$$

Then, we have (fixed):

$$\begin{aligned} e^{tA} &= \cos(\omega t)I + \frac{\sin(\omega t)}{\omega}A \\ \frac{d}{dt}Ae^{tA} &= -\omega\sin(\omega t)I + \cos(\omega t)A \\ Ae^{tA} &= \cos(\omega t)A + \frac{\sin(\omega t)}{\omega}A^2 \end{aligned}$$

4 Euler's Method

Strain says this is like the magic wand, where if we know how to use it, we can solve anything. In the initial value problem, we have some specified data, given at 0. The idea behind the IVPs is that we can forget all the history in the past time and just look at the current state at any given point in time.
Consider the Initial Value Problem:

$$y'(t) = f(t, y(t)) \qquad y(t_n) = y_n$$

The integral equation of this is:

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \int_{t_n}^{t_{n+1}} y'(s)\ ds \\ &= y(t_n) + \int_{t_n}^{t_{n+1}} f(s, y(s))\ ds \end{aligned}$$

We conclude that we should integrate from t_n to t_{n+1}. We know how to perform numerical integration in many ways, so it's simply a matter of choosing one.
How to approximate the integral? We can take, for example, 5 points, with ECTR: $w_0, \ldots, w_4$:

$$\int_{t_n}^{t_{n+1}} f(s, y(s))\ ds = w_0 f(t_n, y_n) + w_1 f(t_{n+\frac{1}{4}}, y_{n+\frac{1}{4}}) + \cdots + w_4 f(t_{n+1}, y_{n+1})$$

Actually, we can't use ECTR because we don't know things like $y_{n+\frac{1}{4}}$. Instead, we want to find some numerical integration rule that only uses the endpoints (namely only the left endpoint), because we want an explicit formula for moving forward.
This is Euler's method:

$$\begin{aligned} u_{n+1} &= u_n + hf(t_n, u_n), \qquad u_0 = y_0 \\ \int_{t_n}^{t_{n+1}} f(s, y(s))\ ds &= hf(t_{n+1}, y_{n+1}) \end{aligned}$$

This is called "Backwards Euler", and an implicit equation,

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}).$$

This was thought to be 'backwards' but turns out to be incredibly useful.
Trapezoidal Rule: Implicit. Alternatively, we can use trapezoidal rule even without endpoint correction (with just a simple trapezoid, averaging the two values):

$$\sum_{t_n}^{t_{n+1}} f(s, y(s))\ ds = \frac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right]$$

Midpoint Estimation: Implicit. Also, alternatively, we can take the midpoint in the domain and estimate the function value.

$$\begin{aligned} \int_{t_n}^{t_{n+1}} f(s, y(s))\ ds &= hf(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}) \\ &= hf\left(t_n + \frac{h}{2}, \frac{y_n + y_{n+1}}{2}\right) \end{aligned}$$

Gaussian Integration: (w_j, θ_j) on $[-1, 1]$

$$\int_{t_n}^{t_{n+1}} f(s, y(s))\ ds = \frac{h}{2} \sum_{p=0}^{w} \left[w_p f \left(t_n + \frac{h}{2} + \frac{h}{2}\theta_p, \underbrace{y\left(t_n + \frac{h}{2} + \frac{h}{2}\theta_p \right)}_{} \right) \right],$$

where we leave these guys for later, because we don't have a way of computing these yet, until Runge-Kutta methods.
Suppose we decide that actually we can use some integration rule with:

$$\int_{t_n}^{t_{n+1}} f(s, y(s))\ ds = h\left[\alpha f(t_n, y_n) + \beta f(t_{n+1}, y_{n+1}) + \gamma f(t_{n-1}, y_{n-1}) + \cdots + \delta f(t_{n-17}, y_{n-17})\right]$$

This is the idea that when we integrate, we can use past values (and not future). We call this "Multistep"
We know the exact solution:

$$\begin{aligned} y_{n+1} &= y_n + \int_{t_n}^{t_{n+1}} y'(s)\ ds \\ &= y_n + hf(t_n, y_n) - \underbrace{hf(t_n, y_n) \int_{t_n}^{t_{n+1}} y'(s)\ ds}_{h\tau_n} \end{aligned}$$

Euler's method says to do something a bit different, to get an approximation to the integral:

$$u_{n+1} = u_n + hf(t_n, u_n).$$

Ideally, we want to make these equations look alike and subtract them.
Hence we have the **local truncation error**, where τ stands for **t**runcation

$$\tau_n := \frac{y_{n+1} - y_n}{h} - f(t_n, y_n)$$

Notice that we take Euler's method and apply it to the **exact** solution, because it will not exactly satisfy the Euler's method (whereas the meshpoints from our computed solution would not do the trick).

Remark: Another way to express local truncation error is "the residual of exact y in the numerical method".

Notice in the above equation, we have $f(t_n, y_n) = y'_n$, so:

$$\begin{aligned} \tau_n &= \frac{y_{n+1} - y_n}{h} - y'_n \\ &= \frac{y_n + hy'_n + \frac{1}{2}h^2 y''(\xi_n) - y_n}{h} - y'_n, \end{aligned}$$

and

$$\tau_n = \frac{1}{2} hy''(\xi_n) = O(h)$$

so $\tau_n \to 0$ as $h \to 0$. We call this 'consistency', in the sense that our numerical method **converges** to the differential equation. However, the goal is to have the **solution** of the numerical method to converge to the solution of the differential equation.

5 Error Analysis

$$\begin{aligned} & y_{n+1} = y_n + hf(t_n, y_n) + h\tau_n \\ -[\quad & u_{n+1} = u_n + hf(t_n u_n) \quad] \end{aligned}$$

Subtracting these 'remarkably similar-looking' equations gives:

$$e_{n+1} := y_{n+1} - u_{n+1} = [y_n - u_n] + h\left(f(t_n, y_n) - f(t_n, u_n)\right) + h\tau_n$$

We cry a bit because our error does not decrease. But recognize that we march along step by step, so there's no reason for our error to decrease. We bound this via triangle inequality:
Assuming f is Lipschitz, not only do we have the existence of a solution within our relevant timespan, but also we get a bound for the error in our numerical method.

$$|e_{n+1}| \le |e_n| + hL|e_n|$$

For our solution, we needed f differentiable down to 2 derivatives anyway, where

$$\tau_n = \frac{1}{2}hy''(\xi_n) = O(h),$$

from above.
Hence we have a recurrence relation with **inequality** as opposed to equality.

$$|e_{n+1}| \le (1 + hL)|e_n| + h|\tau_n|$$

By the compound interest argument, we know the $(1 + hL)$ grows at a bounded rate. The difference between compounding yearly versus daily versus every minute is bounded. As $h \to 0$, we have more steps; however, at each step, the difference in function values is smaller.
Let $\tau := \max_n |\tau_n| = O(h)$. Then, we have:

$$\begin{aligned} |e_n| &\le (1 + hL)\,|e_{n-1}| + h\tau \\ |e_{n+1}| &\le (1 + hL)^2\,|e_{n-1}| + \left[(1 + hL)^1 + (1 + hL)^0\right] h\tau \\ &\le (1 + hL)^{n+1}\,|e_0| + \underbrace{\left[(1 + hL)^n + (1 + hL)^{n-1} + \cdots + (1 + hL)^0\right]}\, h\tau \end{aligned}$$

We call this bit 'stability' as $h \to 0$ (errors don't explode). Later with consistency (the idea that our method converges to the equation), these two will imply convergence of the numerical solution to the exact solution.
We assume our inital error $|e_0| = 0$, otherwise this error is very expensive. Then the remaining terms is a finite geometric series:

$$\begin{aligned} |e_{n+1}| &\le \frac{(1 + hL)^{n+1} - 1}{(1 + hL) - 1} h\tau \\ &= \frac{1}{L}\left[(1 + hL)^{n+1} - 1\right]\tau \end{aligned}$$

We estimate $(1 + hL)$ by:

$$1 + hL \le 1 + hL + \frac{1}{2!}(hL)^2 + \frac{1}{3!}(hL)^3 + \frac{1}{4!}(hL)^4 + \cdots = e^{hL}$$

Then, we conclude:

$$(1 + hL)^n \leq [e^{hL}]^n = e^{nhL} = e^{t_n L},$$

where $t_n = nh$, because n varies from 0 to τ. Notice this is a sloppy (inoptimal bound) but looks elegant. In particular, our error may decrease due to the stiffness of the equation.
The idea between convergence proof is to know that theoretically we have a solution; later we'll worry about if these are practical.
We conclude that we proved a theorem:

Theorem 5.1. If y'' is bounded with $|y''| \leq M$, and f is Lipschitz, then for $0 \leq t_n \leq T$,

$$|u_n - y_n| \leq \frac{e^{t_n L - 1}}{L} \frac{1}{2} hM,$$

where $n \to \infty$ and $h \to 0$.

Even though we are sampling at more and more points, our maximum error converges to zero. We call this "convergence" in the sense that the numerical solution converges to the exact solution in a fixed interval $[0, T]$.

Lecture ends here.

6 Office Hour

Review Gaussian Integration:

$$\int_{-1}^{1} f(t)\ dt = \sum_{j=0}^{p} w_j f(t_j),$$

for $p + 1$ points.
This gives

$$\sum_{j=0}^{p} w_j t_j^k = \int_{-1}^{1} t^k\ dt, \qquad 0 \leq k \leq 2p + 1,$$

where we normally have $0 \leq k \leq p$, but the magic in Gaussian Integration is we have $2p + 1$ here.

Math 128A, Summer 2019
Lecture 19, Wednesday 7/24/2019

1 Runge-Kutta method

We open up class answering questions about integration rules.
Last time, we proved that Euler's method converges to the exact solution:

$$u_{n+1} = u_n + hf(t_n, u_n),$$

in that $(u_n - y_n) \to 0$ as $n \to \infty, h \to 0$, with
The idea behind Runge-Kutta methods is to be like Euler but get a higher order of accuracy.
Euler's method says that for u_n, if this is a solution to the diff eq (which it has no hope being), then the slope throughthe point u_n would be $f(t_n, u_n)$. Euler's method is to extend that slope in a line, and extrapolate to the point at t_{n+1}, then repeat for t_{n+1} to t_{n+2}.
Alternatively, consider taking $v := u_n + hf(u_n)$, the approximate solution at the end of some interval. Then we suddenly want to use the trapezoidal rule, $u_{n+1} := u_n + \frac{h}{2}(f(u_n) + f(v))$. We call this the 'explicit trapezoidal rule (ETR)', where now we know v and $f(v)$.
Another way is to use Euler's to get halfway to get an approximate solution, then to set:

$$v = u_n + \frac{h}{2}\underbrace{f(u_n)}_{k_1}$$
$$u_{n+1} = u_n + h\underbrace{f(v)}_{k_2}$$

These all follow the **Runge-Kutta framework**. That is, we define k_1, k_2 as:

$$k_1 = f(u_n + h[a_1k_1 + \cdots + a_sk_s])$$
$$k_2 = f(u_n + h[a_1'k_1 + \cdots a_s'k_s])$$

We take a linear combination of the slopes to get to the next step. Essentially, we take two steps forward to preview how it looks, then we actually take one step.
Everything depends on the A matrix, $A = [a_{ij}]$, and the vector $b = [b_i]$. Our hope is for linear algebra to save the day.
The standard way of describing Runge-Kutta methods is called the 'Butcher' array, where we put A above b^T.
For example, for Euler's method:

$$k_1 = f(u_n)$$
$$u_{n+1} = u_n + hk_1.$$

Here, we have $A = 0$ (the zero matrix), and b is just 1.
Last time, we also talked about **backwards Euler**, which is an implicit method, where we look at the solution at the end of an interval. Because

$f(u_{n+1})$ is not in our desired form of h times something, we deduce that $f(u_{n+1})$ MUST be k_1.

$$u_{n+1} = u_n + h\underbrace{f(u_{n+1})}_{k_1},$$

so we write k_1 in terms of itself:

$$k_1 = f(u_n + hk_1)$$

As an aside, we can try to solve this via bisection or fixed-point iteration. We take k_1 and shrink it down dramatrically by h, so there is high probability that this will be a contractive iteration. Then, we calculate:

$$u_{n+1} = u_n + hk_1.$$

We can read off the Butcher array, where our 1×1 matrix $A = [1]$, and our array is $b = [1]$.
Trapezoidal Rule:

$$u_{n+1} = u_n + \frac{h}{2}\left(\underbrace{f(u_n)}_{k_1} + \underbrace{f(u_{n+1})}_{k_2}\right)$$

Hence we have:

$$\begin{aligned} k_1 &= f(u_n) \\ k_2 &= f\left(u_n + \frac{h}{2}\left[k_1 + k_2\right]\right) \end{aligned}$$

Then, our Butcher array is simply: $[0, 0; 1/2, 1/2; 1/2, 1/2]$.
For **ETR**, we have:

$$\begin{aligned} k_1 &= f(u_n) \\ k_2 &= f(u_n + hk_1), \end{aligned}$$

which nets us a Butcher array of $[0, 0; 1, 0; 1/2, 1/2]$.
For **Midpoint**, we have:

$$\begin{aligned} u_{n+1} &= u_n + hf\left(\frac{u_n + u_{n+1}}{2}\right) \\ k_1 &= f\left(\frac{u_n + u_{n+1}}{2}\right) \\ u_{n+1} &= u_n + hk_1, \end{aligned}$$

where

$$\begin{aligned} \frac{u_n + u_{n+1}}{2} &= \frac{u_n + u_n + hk_1}{2} \\ &= u_n + \frac{h}{2}k_1 \end{aligned}$$

which gives

$$k_1 = f\left(u_n + \frac{h}{2}k_1\right),$$

which is k_1 expressed in terms of k_1 as desired. For this 1-stage Runge-Kutta method, we write the Butcher array $[1/2; 1]$.
Break time.
So we have a general Runge-Kutta method with :

$$k_i = f\left(u_n h \sum_{j=1}^{s} a_{ij} k_j\right), \qquad 1 \le i \le s$$
$$u_{n+1} = u_n + h \sum_{i=1}^{s} b_i k_i.$$

How can we make this method accurate?
Recall before we looked at the local truncation error,

$$\begin{aligned}\tau &= \frac{y_{n+1} - y_n}{h} - f(y_n) \qquad \text{Euler} \\ &= \frac{y_{n+1} - y_n}{h} - \frac{1}{2}\left(f(y_n) + f(y_{n+1})\right)\end{aligned}$$

Hence for Runge-Kutta methods, everything depends on the local truncation errors.

$$\tau = \frac{y_{n+1} - y_n}{h} - \sum_{i=1}^{s} b_i k_i =: O(h) \qquad \text{when?},$$

when

$$k_i = f\left(y_n + h \sum_{j=1}^{s} a_{ij} k_j\right)$$

Because τ is a function of h, we can Taylor expand it :

$$\tau = \tau(h=0) + \tau'(0)h + \frac{\tau''(0)}{2!}h^2 + \cdots + \frac{\tau^{(p)}(0)}{p!}h^p + \cdots,$$

where $\tau(0) = \tau'(0) = \tau^{(p-1)}(0) = 0$.
What is $\tau(0)$?

$$\begin{aligned}\tau(h) &= \frac{y_n + hy_n' + \frac{1}{2}h^2 y''(\xi_n) - y_n}{h} - \sum_{i=1}^{s} b_i k_i(h) \\ &= y_n' + \frac{1}{2}hy''(\xi_n) - \sum_{i=1}^{s} b_i k_i(h),\end{aligned}$$

as $h \to 0$. Hence we need to know what k_i approaches as $h \to 0$.

$$k_i = f(y_n + h \sum_{j=1}^{s} a_{ij} k_j) \to f(y_n),$$

where $f(y_n) \equiv f(y_n')$. Hence we write, as $h \to 0$,

$$\tau(h) \to f(y_n) - \left(\sum_{i=1}^{s} b_i\right) f(y_n) = \left(1 - \sum_{i=1}^{s} b_i\right) f(y_n),$$

and we conclude that to get first order of accuracy, we require:

$$\sum_{i=1}^{s} b_i := 1.$$

Now for **second order** of accuracy, we find:

$$\tau(h) = y_n' + \frac{1}{2}hy_n'' + \frac{1}{3!}h^2 y_n''' + \cdots - \left(\sum_{i=1}^{s} b_i\right)\left[k_i(0) + k_i'(0)h + \frac{1}{2!}k_i''(0)h^2 + \cdots\right].$$

Hence we conclude our 2nd order condition is:

$$\frac{1}{2}y_n'' - \sum_{i=1}^{s} b_i k_i'(0) = 0$$

and our 3rd order condition is:

$$\frac{1}{3!}y_n''' - \sum_{i=1}^{s} b_i k_i''(0) = 0.$$

So we write:

$$\begin{aligned} y' &= f(y) \\ y'' &= f'(y)y' = f'(y)f(y) \\ y''' &= f''(y)(y')^2 + f'(y)y'' \\ &= f''(y)f(y)^2 + f'(y)f'(y) + f(y) \\ y''' &= f''f^2 + (f')^2 f. \end{aligned}$$

This is one source of the combinatorial explosion that makes it hard, say, to find the 10th order Runge Kutta method.
Key conclusions from above are:

$$f'(y)f(y) = y''$$

and

$$y''' = f''(y)[f(y)]^2 + [f'(y)]^2 f(y).$$

We have two terms, and when we differentiate, we will have two separate conditions.
Now we calculate k_i:

$$\begin{aligned} k_i &= f\left(y_n + h\sum_j a_{ij}k_j\right) \\ k_i'(h) &= f'\left(\underbrace{y_n + h\sum_j a_{ij}k_j}\right)\left(\sum_j a_{ij}k_j + \sum_j a_{ij}k_j'\right), \end{aligned}$$

where by chain rule, we have to remember that we have k_j is a function on h, and hence we have the right factor. As $h \to 0$, we have:

$$\begin{aligned} k_i' &= f'(y_n) \sum_j [a_{ij} k_j(0)] \\ &= \left(\sum_j a_{ij} \right) f'(y_n) f(y_n) \end{aligned}$$

Notice $\sum_j a_{ij} k_j$ is bounded as $k_i(0) = f(y_n)$ as $h \to 0$, which lets us use the above convergences.
So to satisfy the 2nd order condition,

$$\frac{1}{2} f'(y_n) f(y_n) - \sum_{i=1}^{s} b_i \sum_{j=1}^{s} a_{ij} f'(y_n) f(y_n) = 0,$$

so we have:

$$\begin{aligned} \frac{1}{2} &= \sum_{i=1}^{s} b_i \left(\sum_{j=1}^{s} a_{ij} \right) \\ \implies b^T A e &= \frac{1}{2}, \end{aligned}$$

where $e := [1; 1; 1; \cdots 1]$ and

$$(Ae)_i := \sum_{j=1}^{s} a_{ij} 1$$

Differentiating further and going straight $h \to 0$, we have:

$$\begin{aligned} k_i'(h) &= f' \left(\underbrace{y_n + h \sum_j a_{ij} k_j}_{} \right) \left(\sum_j a_{ij} k_j + \sum_j a_{ij} k_j' \right) \\ k_i''(0) &= f''(y_n) \left(\sum_j a_{ij} f(y_n) \right)^2 + f'(y_n) \left[\sum_j a_{ij} f'(y_n) \sum_l a_{jl} f(y_n) \right] \end{aligned}$$

So we have two conditions:

$$\begin{aligned} \frac{1}{3!} &= \sum_i b_i \left(\sum_j a_{ij} \right)^2 \\ \frac{1}{3!} &= 2 \sum_i b_i \sum_j a_{ij} \sum_l a_{jl} \end{aligned}$$

The first equation is with matrices, but we don't necessarily use linear algebra; however, if we wish, we can set the second equation equal to $b^T A^2 e$, as defined before.

Lecture ends here.

Tomorrow we'll build Runge-Kutta methods with p^2 stages in order p.

Math 128A, Summer 2019

Lecture 20, Thursday 7/25/2019

1 Review

Strain opens with an example of how we read off a Butcher array. Suppose our roommate 'discovers' a new Runge-Kutta method, say:

$$\begin{aligned} u_{n+1} &= u_n + h\left[\frac{2}{3}\underbrace{f}_{k_3}\left(u_n + \frac{h}{3}\underbrace{f}_{k_2}\left(u_n + \frac{h}{3}\underbrace{f}_{k_1}(u_n)\right)\right) + \frac{1}{3}\underbrace{f}_{k_4}(u_{n+1})\right] \\ &= u_n + h\left(\frac{2}{3}k_3 + \frac{1}{3}k_4\right) \end{aligned}$$

WLOG we number the k's in the order, although the Butcher array could easily be permuted.
The butcher array here is

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix}$$

Notice that the last row of the butcher array (b) is the same as the last row of the matrix.
Also notice the first step of the new step is the same as the last step of the previous step, so we don't actually take 4 computations, only 3. We call this 'First same as last'.
We write down the Butcher array so we can check the order conditions. It helps us not have to taylor expansion the f's a ton of times.
Recall from yesterday's lecture, the first (order) condition is:

$$\sum b_i = 1$$

and the second (order) condition is:

$$b^T A e = \frac{1}{2}$$

$c = Ae$ is simply the row-sums of the Butcher matrix; in our example above, we have:

$$c = Ae = \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ 1 \end{bmatrix}$$

Then we check second-order accuracy:

$$b^T Ae = \begin{bmatrix} 0 & 0 & \frac{2}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ 1 \end{bmatrix}$$

We only have first-order accuracy. But we invent reasons why this is better than existing methods, such as if we run on some trinary computer (as opposed to binary), our method would be exact.

2 Introduction: Deferred Correction

We used to talk about both extrapolation and deferred correction, but we found out that extrapolation isn't very good.
arbitrary-order
The idea is that given a low-order Runge-Kutta method for

$$y' = f(t, y) \qquad \to \qquad u_n \ \text{ approx solution}$$

that we derive an ODE for the error. But before this, we need a way to go from u_n to $u(t)$. To do this, we interpolate to get a differentiable function, $u(t)$.

Then, we get an ODE for error

$$e(t) = y(t) - u(t).$$

Next we solve by low-order method for e_n. Finally, we correct our solution given our found error:

$$u_n^1 = u_n + e_n$$

Hence we have a 'deferred' correction where we have an approximate solution, then subtract out the error at the end.
Essentially, we apply a low-order method (say first-order) to each of the solution and error, yielding a higher-order (second-order) result. We can iterate back up to Interpolation to the end, repeatedly.

2.1 Autonomizing: Suppressing t

$$\begin{aligned} y' &= f(t, y) \\ Y' := \begin{bmatrix} t \\ y \end{bmatrix}' &= \begin{bmatrix} 1 \\ y' \end{bmatrix} = F(Y) \end{aligned}$$

This is called 'autonomizing' the system, where we do not have t. Compare against $y' = f(t, y)$, which is a non-autonomous system.

3 Euler's Method

(1)

$$u_{n+1} = u_n + hf(t_n, u_n)$$

(2) We have a function and interpolate it with some polynomial $u(t)$ of some degree, say $n+1$ points.
(3) Now we want an ODE for error, $e(t) := y(t) - u(t)$. We know how to differentiate at all the mesh points; that is,

$$\begin{aligned} e'(t) &= y'(t) - u'(t) \\ &= f(t, y(t)) - u'(t) \end{aligned}$$

We rewrite: $e(t) = y(t) - u(t)$ to get $y(t) = u(t) - e(t)$ to get a differential equation from the above:

$$\begin{aligned} e'(t) &= f(t, u(t) - e(t)) - u'(t) \\ &= g(t, e(t)) \end{aligned}$$

Then (4) we use Euler's method:

$$\begin{aligned} e_{n+1} &= e_n + hg(t_n, e_n) \\ &= e_n + h\left[f(t_n, u_n + e_n) - u'(t_n)\right] \end{aligned}$$

and then (5), we have

$$u_n^1 = u_n + e_n,$$

and we ought to have a higher-order accuracy.

3.1 Example of Euler's Method

We work out the smaller details. Suppose we interpolate at points $n, n+1, \ldots, n+p$. We have a bunch of values, so we put a polynomial through it. Actually, we never need the polynomial itself; we need only its derivatives.
(1)

$$u_{n+j+1} = u_{n+j} + hf(t_{n+j}, u_{n+j})$$

For $p = 2$, we have:

$$u(t_{n+j}) = u_{n+j}, \qquad 0 \le j \le p = 2$$

This gives a quadratic polynomial (3 points). We have $j = 0, 1, 2$.
We say:

$$u'(t_{n+j}) = d_{j,0}u_n + d_{j,1}u_{n+1}d_{j,2}u_{n+2}.$$

Let's start with the best initial value and not correct it, only correcting our subsequent values (we accept the starting error and want to move forward as opposed to correcting the past). Hence:

$$\begin{aligned} e_n &:= 0 \\ e_{n+1} &= e_n + h\left[\overbrace{f(t_n, u_n + e_n)}^{d_0} - u'(t_n)\right] \\ e_{n+2} &= e_{n+1} + h\left[\underbrace{f(t_{n+1}, u_{n+1} + e_{n+1})}_{d_1} - u'(t_{n+1})\right] \end{aligned}$$

This gives:

$$\begin{aligned}
u^1_{n+1} &= u_{n+1} + e_{n+1} \\
u^1_{n+2} &= u_{n+2} + e_{n+2} \\
&= u_{n+1} + hf(t_{n+1}, u_{n+1}) + e_{n+1} + h\left[f(t_{n+1}, u_{n+1} + e_{n+1}) - u'(t_{n+1})\right] \\
&= u_n + hf(t_n, u_n) + hf(t_{n+1}, u_{n+1}) + h\left[f(t_n, u_n) - u'(t_n)\right] \\
&\quad + h\left[f\left(t_{n+2}, u_{n+1} + h\left[f(t_n, u_n) - u'(t_n)\right]\right) - u'(t_{n+1})\right]
\end{aligned}$$

where u^1_{n+2} is the final corrected second-order solution u_{n+2} at $t_{n+2} = t_n + 2h$.
We define the k's:

$$\begin{aligned}
k_1 &:= f(t_n, u_n) \\
k_2 &:= f(t_{n+1}, u_n + hk_1) \\
k_3 &:= f\left(t_{n+1}, u_n + hk_1 + h\left(k_1 - u'(t_k)\right)\right) \\
&= f\left(t_{n+1}, 2hk_1 - hu'(t_n)\right)
\end{aligned}$$

So we have:

$$\begin{aligned}
u'(t_n) &= d_{0,0}u_n + d_{0,1}u_{n+1} + d_{0,2}u_{n+2} \\
&= \left(d_{0,0} + d_{0,1} + d_{0,2}\right)u_n \\
&\quad + (d_{0,1} + d_{0,2})hf(t_n, u_n) \\
&\quad + d_{0,2}hf(t_{n+1}, u_{n+1})
\end{aligned}$$

but $d_{0,0} + d_{0,1} + d_{0,2} = 0$ because these are the coefficients of the result after differentiating. We define (as we did for Euler-Maclaurin):

$$\begin{aligned}
d_{0,0} &= \frac{-3}{2} \\
d_{0,1} &= 2 \\
d_{0,2} &= -\frac{1}{2}
\end{aligned}$$

Then this gives:

$$k_3 = f\left(t_{n+1}, u_n + h\left[(2 - d_{0,1} - d_{0,2})k_1 + d_{0,2}k_2\right]\right)$$

and by the same tokens as above, we have:

$$\begin{aligned}
u_{n+2} &= u_n + hk_1 + hk_2 + h(k_1 - u'(t_n)) + h[d_3 - u'(t_{n+1})] \\
&= u_n + hk_1 + hk_2 + hk_1 - (d_{0,1} + d_{0,2})h^2k_1 - d_{0,2}h^2k_2 \\
&\quad + hk_3 - (d_{1,1} + d_{1,2})h^2k_1 - d_{1,2}h^2k_2
\end{aligned}$$

We read off the Butcher array:

$$\begin{bmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
2 - h(d_{0,1} + d_{0,2}) & -hd_{0,2} & 0 \\
2 - h(d_{0,1} + d_{0,2}) - h(d_{1,1} + d_{1,2}) & 1 - hd_{0,2} - hd_{1,2} & 1
\end{bmatrix}$$

Because this is an explicit scheme, we have zeroes on and above the diagonal.

Recall that we need to divide by two at the end (to standardize), because we are taking a step size $2h$, and the definition of the Butcher array uses a step size of h.
The point of the Butcher array is that the numbers within are independent of f, u, h.
Letting $d_{1,0} := \frac{-1}{2}, d_{1,1} := 0, d_{1,2} := \frac{1}{2}$, as we normally do for divided difference, we have our butcher array:

$$\begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Checking second-order accuracy, we have:

$$\sum b_i = 0 + \frac{1}{2} + \frac{1}{2} = 1$$
$$\sum b_i c_i = 0 \cdot 0 + \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2},$$

so we conclude we have second-order accuracy.
Breaktime.

4 (Iterated) Deferred Correction

We have a first-order solution,

$$\begin{aligned} u^0_{n+j+1} &= u^0_{n+j} + hf(t_{n+j}, u^0_{n+j}), \qquad 0 \le j \le p-1 \\ u^0_n &= u_n \\ \implies u_n(t_{n+j}) &= u_{n_j} \\ e_{n+j+1} &= e_{n+j} + h[f(t_{n+j}, u_{n+j} + e_{n+j}) - u'_n(t_{n+j})] \end{aligned}$$

We define our corrected solution to be:

$$\impliedby u^1_{n+j} = u^0_{n+j} + e_{n+j}, \qquad 1 \le j \le p$$

The limit to this iterative scheme is the interpolating polynomial; at some point, the interpolation will no longer correct anything. This iterative scheme $u^k \mapsto u^{k+1}$ looks like fixed point iteration.
Our first question is will this fixed-point iteration converge? Our zeroth question is if this converges, what will it net us?
We notice that if we have convergence, our error at each step (to be corrected) is zero; hence $e_{n+j+1} = e_{n+j} = 0$, hence the rest of the equation must be zero:

$$0 = h[f(t_{n+j}, u_{n+j} + e_{n+j}) - u'_n(t_{n+j})],$$

and at the limit, we have:

$$f(t_{n+j}, u_{n+j}) - u'_n(t_{n+j}) = 0, \qquad 0 \le j \le p-1$$

Remark: This is to say that the interpolating polynomial exactly satisfies the differential equation at the interpolating points.

We let d_i be δs times h, and we get:

$$\begin{aligned} hu'_n(t_{n+j}) &= d_{j,0}u_n + d_{j,1}u_{n+1} + \cdots + d_{j,p}u_{n+p} \\ hf(t_{n+j}, u_{n+j}) &= d_{j,0}u_n + \cdots + d_{j,p}u_{n+p}. \end{aligned}$$

This is a (fully) implicit scheme with p stages (where dependencies require us to solve everything at once), but our iterated scheme has p^2 stages; however, it is explicit.
Our second line is a Runge-Kutta method; however, it may not look like it. It expresses the stages in terms of the new values, whereas usually we express values in terms of the new stages. Define $k_j := f(t_{n+j}, u_{n+j})$. To get around this,

$$\begin{aligned} hk_j &= d_{j,0}u_n + \cdots + d_{j,p}u_{n+p} \\ &= d_{j,0}u_n + d_{j,1}(u_{n+1} - u_n + u_n) + \cdots + d_{j,p}(u_{n+p} - u_n + u_n) \\ &= d_{j,1}(u_{n+1}, u_n) + d_{j,2}(u_{n+2} - u_n) + \cdots + d_{j,p}(u_{n+p} - u_n), \end{aligned}$$

and this gives us:

$$\begin{aligned} h \begin{bmatrix} k_1 \\ \vdots \\ k_p \end{bmatrix} &= \begin{bmatrix} d_{1,1} & \cdots & d_{1,p} \\ \vdots & & \vdots \\ d_{p,1} & \cdots & d_{p,p} \end{bmatrix} \begin{bmatrix} u_{n+1} - u_n \\ \vdots \\ u_{n+p} - u_n \end{bmatrix} \\ hk &= D(u_{n+j} - u_n) \end{aligned}$$

and let $C := D^{-1}$, which then gives us:

$$\begin{bmatrix} u_{n+1} - u_n \\ \vdots \\ u_{n+p} - u_n \end{bmatrix} = hC \begin{bmatrix} k_1 \\ \vdots \\ k_p \end{bmatrix},$$

which looks a lot more like a Runge-Kutta method.

Remark: So in summary, iterated differed correction is a way to solve fully implicit Runge-Kutta methods.

Proving convergence of Runge-Kutta methods is incredibly difficult, so we skip over the proof.

Next week, we'll talk about stiff equations and multistep methods.

Math 128A, Summer 2019
Lecture 21, Monday 7/29/2019

1 Review, Homework 5

1.1 Problem 2

The idea behind a difference equation is a lot simpler than a differential equation but is very parallel. Recall the problem is to find the general solution of

$$u_{j+2} = u_{j+1} + u_j$$

The space of equations is two-dimensional. There are two ways to approach this, and we do it the better way of the two.

First consider the separate problem:

$$\begin{aligned} y''' &= f(t, y, y', y'', z) \\ z' &= g(t, z) \end{aligned}$$

To solve this sort of problem, usually this is difficult to solve; however, for numerical purposes, we want to systematically convert this to a first-order system (which we know works). First, we want to convert the variables (with different levels of differentiation) into new variables. We can make this an autonomous system:

$$u' := \begin{bmatrix} t \\ y \\ y' \\ y'' \\ z \end{bmatrix}' = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}' = \begin{bmatrix} 1 \\ u_2 \\ u_3 \\ f(u_0, u_1, u_2, u_3, u_4) \\ g(u_0, u_4) \end{bmatrix} =: F(u)$$

Remember the idea behind (applied) mathematics is not to solve problems but to move problems around to somewhere we already have a big hammer to drop onto it (we already know how to solve first-order equations).

Remark: Technically our construction is dimensionally off, so we'll want to insert standardizing factors to match the dimensions of y:

$$u' = \begin{bmatrix} y_0 t/h \\ y \\ hy' \\ \frac{h^2}{2!} y'' \\ z \end{bmatrix}$$

So back to our original problem, we construct:

$$U_j = \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} U_{j+1} = \begin{bmatrix} u_{j+1} \\ u_{j+2} \end{bmatrix} = \begin{bmatrix} u_{j+1} \\ u_{j+1} + u_j \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_j \\ u_{j+1} \end{bmatrix} =: FU_j = F^{j+1} U_0$$

1.2 Question 5

Suppose $y(t)$ is the exact solution to the following initial value problem:

$$\begin{aligned} y' &= f(t, y(t)) \\ y(0) &= y_0 \\ e'(t) &= f(t, u(t) + e(t)) - u'(t) \quad \text{(we use this)} \\ f(y) &= \lambda y \end{aligned}$$

Because we have $f(t, y(t)) = y' = \lambda y$, we conclude $y(t) = e^{t\lambda} y(0)$ (this can be easily verified).

$$\begin{aligned} e' &= \lambda(u + e) - u' \\ &= \lambda e + \lambda u - u' \\ (u + e)' &= \lambda(u + e) \\ u + e &= e^{t\lambda} \left[u(0) + e(0)\right] = y(t), \end{aligned}$$

and we are done.

2 Homework: euler.m

There is no iteration within Euler's method; our step is simply from t to $t + 1$. The function is 4 in, 4 out.
Consider the Matlab function:

```
function v = f(t,u,p)
  v(1) = 0.7/(u(1)**2 + u(2)**2)^1.5
  v(2) = -0.3/(u(1)**2 + u(2)**2)^1.5
  v(3) = u(3)
  v(4) = u(4)
end
```

We put 4 in and get 4 out. We call this via `euler(a,b,y0,@f,p)`.

3 Multistep Methods

We haven't said everything about Runge-Kutta methods (that would take a few semesters), but we'll proceed past this.

The idea behind multistep methods is to use previous information and essentially be **unlike Runge-Kutta methods** (where we take samples and slope estimates in between t_n and t_{n+1}). In Runge-Kutta, we don't look backwards (i.e t_{n-17}). However, if we assume that historical data is indicative of the future, we may want to take Multistep methods into account.

To be more specific, suppose we have a bunch of points: $t_{n+1}, t_n, \ldots, t_{n-k+1}$. We write it this way because if $k = 1$, then this yields a "one-step" method, and $k > 1$ yields a "k-step methods" . Hence Runge-Kutta methods are 1-step methods. One obvious idea is to take the values of y at $t_{n-k+1}, \ldots, t_n$ (and possibly t_{n+1} which we don't know) and interpolate per our usual integral equation:

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \int_{t_n}^{t_{n+1}} y'(s)\ ds \\ P(t_{n-j}) &= y'(t_{n-j}) \end{aligned}$$

We have two possibilities:
(1) $0 \leq j \leq k-1$ yields an **explicit** method as it does not use any unknown values. We call this a k-step Adams method.
(2) $-1 \leq j \leq k-1$ yields an **implicit** k-step Adams method.

Example: Implicit 2-step Adams Method:
Consider the following:

$$\begin{aligned} p(t_{n+1}) = y'(t_{n+1}) &\cong f(t_{n+1}, u_{n+1}) \\ p(t_n) = y'(t_n) &\cong f(t_n, u_n) \end{aligned}$$

We can write:

$$\begin{aligned} \int_{t_n}^{t_{n+1}} P(s)\ ds &= \int_{t_n}^{t_{n+1}} \frac{s - t_{n+1}}{t_n - t_{n+1}} f_n + \frac{s - t_n}{t_{n+1} - t_n} f_{n+1}\ ds \\ &= \frac{1}{2}\left(t_{n+1} - t_n\right) f_n + \frac{1}{2}\left(t_{n+1} - t_n\right) f_{n+1} \\ &= b_n \left[\frac{1}{2} f_n + \frac{1}{2} f_{n+1}\right] \end{aligned}$$

This 2-step Adams method yields exactly equivalent to the Trapezoidal rule. For multistep methods, we always want to keep open the possibility that the length of the step h changes at each step. The reason is that to get these multi-step methods, we start with a single method. If every step is the same length, then we start off with way too much error from our first step. A better idea would be to take a very tiny step for our 1-step first step, then gradually larger steps as we go forward.

We have

$$\begin{aligned} p(s) &= \sum_{j=0}^{k-1} L_j(s) f_{n-j} \quad \text{(explicit)} \\ \implies u_{n+1} &= u_n + \int_{t_n}^{t_{n+1}} p(s)\ ds \\ &= u_n + \int_{t_n}^{t_{n+1}} \sum_{j=0}^{k-1} L_j(s) f_{n-j}(s)\ ds \\ &= u_n + \sum_{j=0}^{k-1} p_j(s) f_{n-j}(s); \qquad p_j := \int_{t_n}^{t_{n+1}} L_j^p(s)\ ds \end{aligned}$$

And of course, p_j changes with each step, so we have to re-do this last line calculation at each step (accurately and efficiently). One such way is to write out the equations and perform explicit calculations (Gauss used Taylor expansions). Later, people didn't like starting with Taylor expansions, so they use Runge-Kutta methods. The idea is to use Runge-Kutta to start off the first 10 or so steps, then proceed with multi-step methods.
On the other hand, for Implicit methods:

$$\begin{aligned} q(s) &= \sum_{j=-1}^{k-1} L_j(s) f_{n-j} \\ q_j &= \int_{t_n}^{t_{n+1}} L_j(s)\ ds \end{aligned}$$

This gives us:

$$\begin{aligned} L_j^p(s) &= \prod_{l=0, l\neq j}^{k-1} \frac{s - t_{n-l}}{t_{n-j} - t_{n-l}}, \qquad \deg\ k \\ L_j^q(s) &= \prod_{l=-1, l\neq j}^{k-1} \frac{s - t_{n-l}}{t_{n-j} - t_{n-l}}. \qquad \deg\ k+1 \end{aligned}$$

We don't really want to integrate these guys, so we recall we have Gaussian integration with m points evaluates $\int p$ exactly if $\deg(p) \leq 2m-1$. Recall in our homework, with a degree-5 polynomial, we can get the weights exactly and integrate exactly for degree up to 9.

If the degree of the Gaussian integration is high, numerically when we are evaluating the Legendre polynomials, we accumulate a lot of error. What Strain does for his own projects is store the points and weights for distinct Gaussian quadratures in different files and pull them in when needed.

If we look at books covering Multistep methods, we take a lot of time to explain how to exactly calculate the coefficients q_j, L_j. Strain reminds us (as it is not mentioned anywhere) to simply use Gaussian Quadrature when it delivers an exact result.

3.1 Questions on Multistep Methods: Order of Accuracy (Local Truncation Error), Stability, Getting Started

Before we talk about how to get started, we want to first talk about the accuracy of the scheme. We know

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} y'(s)\ ds$$

where y is the exact solution. To compare, we replace y' with the interpolation polynomial at a bunch of points:

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + \int_{t_n}^{t_{n+1}} \left[y'(s) - p(s) + \underline{p(s)}\right]\ ds \\ &= y(t_n) + \sum_{j=0}^{k-1} p_j f(t_{n-j}, y_{n-j}) + \underbrace{\int_{t_n}^{t_{n+1}} y'(s) - p(s)\ ds}_{h\cdot\tau,\text{ local truncation error}} \end{aligned}$$

It turns out that $y'(s)-p(s)$ is not very difficult to estimate, because p is the interpolating polynomial for y'. Interpolating at k points, for $0 \le j \le k-1$, degree $p = k-1$ (explicit Adams), we have:

$$y'(t_{n-j}) = p(t_{n-j}),$$

so

$$y'(s) - p(s) = \frac{y^{(k+1)}(\xi)}{k!}\left[(s-t_n)(s-t_{n-1})\cdots(s-t_{n-k+1})\right]$$

Integrating both sides and dividing by $\frac{1}{h_n}$ to get the actual local truncation error, we have:

$$\int_{t_n}^{t_{n+1}} y'(s) - p(s) = \int_{t_n}^{t_{n+1}} \frac{y^{(k+1)}(\xi)}{k!}\left[(s-t_n)(s-t_{n-1})\cdots(s-t_{n-k+1})\right]\ ds$$

We can use the MVT for integrals:

$$\frac{1}{b-a}\int_a^b f(x)g(x)\ dx = f(\xi)\frac{1}{b-a}\int_a^b g(x)\ dx$$

The key hypothesis (besides continuity and differentiability is $g(x) > 0$. For our original integral above, we only run into trouble at the left endpoint, where $(s-t_n)$ gives $(t_n - t_n) = 0$, and all other points are in the past. We claim we're good to proceed.
Hence we write:

$$\int_{t_n}^{t_{n+1}} y'(s) - p(s) = \frac{y^{(k+1)}(\xi)}{k!}\underbrace{\left[\frac{1}{h_n}\int_{t_n}^{t_{n+1}} (s-t_n)\cdots(s-t_{n-k+1})\ ds\right]}_{=P_k},$$

where we name this bracketed expression due to its importance. A Gaussian quadrature scheme would give the constant P_k. Our question is how big is P_k? We claim that if $h_j \le O(h)$, then we should have $P_k = O(h^k)$, and hence

$\tau = O(h^k)$ for a k-step explicit method.
The local-truncation error is:
For k-step explicit:

$$\tau_p = \frac{P_k}{k!} y^{(k+1)}(\xi)_P$$

For $(k-1)$-step implicit:

$$\tau_q = \frac{q_k}{k!} y^{(k+1)}(\xi_q)$$

Hence we write:

$$\begin{aligned} q_k \frac{\tau_p - \tau_q}{p_k - q_k} &= q_k \frac{1}{k!} y^{k+1}(\xi) + \cdots \\ &\cong \tau_q \end{aligned}$$

If we compute:

$$\text{explicit} \qquad v_{n+1} = v_n + \sum_{i=0}^{k-1} p_j f_{n-j} + h\tau^{(p)}$$

$$\text{implicit} \qquad u_{n+1} = u_n + q_{-1} f_{n+1} + \sum_{j=0}^{k-1} q_j f_{n-j} + h\tau_n^{(q)}$$

Subtracting these gives:

$$\begin{aligned} v_{n+1} - u_{n+1} &= h_n \tau_n^{(p)} - h_n \tau_n^{(q)} \\ \frac{v_{n+1} - u_{n+1}}{h_n} &= \tau_n^{(p)} - \tau_n^{(q)}, \end{aligned}$$

and we write, for the error estimate:

$$\frac{q_k}{p_k - q_k} \frac{v_{n+1} - u_{n+1}}{h_n} \cong \tau_n^{(q)}.$$

The interesting thing about this scheme is that if we perform both the explicit and implicit schemes (where the latter is a lot of work via bisection, Newton's or fixed point iteration), we can express the error as the difference of the two as above.

Remark: We call this "Predictor-Corrector": we use explicit method (requires two steps) to give an error estimate, order $O(h^k)$ tells us how big our next step should be. We feed our result from the explicit method into the implicit method.

Lecture ends here.

Next time, we'll talk about stiff equations (going back to Runge-Kutta methods because they work better).

4 Office Hours

4.1 Deferred Correction

Recall our scheme was

$$y' = f(y)$$

$$u_{n+j+1} = u_{n+j} + hf(u_{n+j}), \qquad 0 \le j \le p-1$$

We call our interpolation $u(t)$, and we have:

$$e'(t) = f(u+e) - u' \tag{1}$$

The error equation is simply what we add to our approximate solution to get the exact solution. That is,

$$\begin{aligned} (u+e)' &= f(u+e) \\ e(t_n) &= 0 \end{aligned}$$

We use (1) to actually solve the equation. We already know what u is, and therefore we can solve via Euler's method:

$$\begin{aligned} e_{n+j+1} &= e_{n+j} + h\left[f(u_{n+j} + e_{n+j}) - u'(t_{n+j})\right] \\ e_n &:= 0, \qquad 0 \le j \le p-1 \end{aligned}$$

Because Euler's method is first-order accurate, these errors are like the exact errors in the relative sense. To see this, write out the Butcher array to verify the order conditions are satisfied.
Now that we have the estimated error, we want to **deliver** the corrected solution, $1 \le j \le p$:

$$u^1_{n+j} = u_{n+j} + e_{n+j}$$

Notice that we put an interpolating polynomial through u, where u' is simply the differentiation coefficients:

$$u'(t_{n+j}) = \sum_{k=0}^{p} d_{j,k} u_{n+k}$$

We run Euler's through all our points of interest $t_n, \ldots, t_{n+p}$ first before interpolating.

Math 128A, Summer 2019
Lecture 22, Tuesday 7/30/2019

1 Review: Homework

```
1 u_{n+j+1} = u_{n+j} + hf(t_{n+j}, u_{n+j})
2 0 \leq j \leq p - 1
3 e_{n+j+1} = e_{n+j} + h \[ f(t_{n+j}, u_{n+j} + e_{n+j}
4     + e_{n+j}) - u'(t_{n+j}) \]
5 0 \leq j \leq p - 1
6 e_n = 0
7 u_{n+j}^{1} = u_{n+j} + e_{n+j}
```

Then once we have line 7, u_{n+j}, we stick it back into line 3, u_{n+j}. We solve for the error and add it onto our approximation, then solve for error again and repeat. We don't change u_n over the different passes.
Our problem with the orbit is about the natural (perpetual) figure-8 orbit around the earth and the moon.

2 Review Lecture 21: Multi-Step Methods

These may not have the same charm as Runge-Kutta methods, perhaps because they're truly simple. We take:

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} p(s)\, ds,$$

where we have the general practice of replacing things we don't know with things we **do** know, which is the interpolating polynomial $p(s)$. Effectively, we interpolate some polynomial through points $t_{n+1-k}, \ldots, t_n$ and then integrate past our interval, over $[t_n, t_{n+1}]$ which is not illegal and grants us an estimate (k-step explicit scheme).

Alternatively, we can put $q(s)$ which also interpolates at t_{n+1}, so we are integrating within our interpolating interval ($k-1$-step implicit scheme), and this ought to be more accurate even when using the same number of points, $t_{n+2-k}, \ldots, t_n, t_{n+1}$.

$$\begin{aligned} v_{n+1} &= u_n + h_n \sum_{j=0}^{k-1} p_j f_{n-j} \\ u_{n+1} &= u_n + h_n \left[q_{-1} f(t_{n+1}, v_{n+1}) + \sum_{j=0}^{k+2} q_j f_{n-j} \right], \end{aligned}$$

where we explicitize the implicit scheme in the second line by taking the problematic term p_j by using the explicit scheme as a predictor and the implicit scheme as a corrector.

If we feed the exact solution y into the above, we have:

$$y_{n+1} = y_n + h_n \sum_{j=0}^{k-1} p_j y'_{n-j} + h_n \tau^p$$

$$y_{n+1} = y_n + h_n \left[q_{-1} f(t_{n+1}, y_{n+1}) + \sum_{j=0}^{k+2} q_j y'_{n-j} \right] + h_n \tau^q,$$

now with the local truncation errors.
To get this result, Strain ponders:

$$v_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p(s)\ ds = y_{n+1} - h\tau^p$$

$$u_{n+1} = y_n + \int_{t_n}^{t_{n+1}} q(s)\ ds = y_{n+1} - h\tau^q,$$

and subtracting these equations gives

$$\begin{aligned} \frac{v_{n+1} - u_{n+1}}{h} &= \tau^p - \tau^q \\ &= (p_k - q_k)\, h^k \end{aligned}$$

However, subtracting these may return 0, which does not give us any insight into how large each of them are.
This is why we are using one step less than our original explicit scheme, so we are the same order of accuracy from the true solution.
To see that $p_k - q_k \neq 0$, we look at:

$$\int_{t_n}^{t_{n+1}} \underbrace{t - t_{n+1}}_{q} \underbrace{[(t-t_n)\cdots(t-t_{n-k+2})]}_{p(t)}\ dt$$

Hence we conclude:

Remark:

$$q_k h^k = \tau^q = \frac{q_k}{p_k - q_k} \frac{v_{n+1} - u_{n+1}}{h}$$

estimates error in the corrected solution, with step size control.

Suppose we want error $\leq$ tolerance. We just took step $t_n \to t_{n+1}$ and error estimate $E = q_k h^k$.

(1) If $|E| \leq tol$, then we want to increase the next step (we didn't waste computation time, but we may have taken too small a step and consequently may require too many steps; we may not get the solution by the time we need it). We increase by some factor, say

$$\theta := \min \left\{ 1 + \frac{1}{k}, 0.9 \left(\frac{tol}{|E|} \right)^{1/k} \right\}; \qquad h_{n+1} := h_n \theta$$

(2) If $|E| \geq tol$, we must reject our step and re-do that step with a smaller step-size, say $h_{n+1} := \theta h_n$ with:

$$\theta := \max \left\{ 0.1, \left(\frac{tol}{|E|} \right)^{1/k} \right\}$$

If we look at the textbook GGK, we can see that adaptive stepsize control is ridiculously more efficient and better-performing than equidistant. For our purposes, in the interest of time (summer), we will not be asked to programatically implement adaptive stepsize control, as it is difficult to debug. We can get about 9-digit accuracy with an equidistant Adams predictor-corrector.

3 Stability: The Second Piece in Proving Convergence

Let's suppose we have k initial values, $u_0, \ldots, u_{k-1}$ to some degree of accuracy. How do we prove that u_n converges to y_n on some fixed interval of time?

Via extrapolating, we have:

$$u_{n+1} = u_n + h\left[p_0 f_n + \cdots + p_1 f_{n-k+1}\right]$$

As we recall (lecture 18), there are two ingredients: (1) consistency in that $\tau = O(h^k)$ and our method converges to the equation, and (2) stability in that as $h \to 0$, errors don't explode.
We checked already that $\tau_n = O(h^k)$ via:

$$y_{n+1} = y_n + h\left[p_0 f(t_n, y_n) + \cdots + p_{k-1} f(t_{n-k+1}, y_{n-k+1})\right] + h\tau_n$$

and additionally for u,

$$u_{n+1} = u_n + h\left[p_0 f(t_n, u_n) + \cdots + p_{k-1} f(t_{n-k+1}, u_{n-k+1})\right] + h\tau_n$$

and subtracting these,

$$|e_{n+1}| \le |e_n| + h\left(|p_0|\underbrace{|f(y_n) - f(u_n)|}_{L|y_n - u_n| = L|e_n|} + \cdots + |p_{k-1}|\ \underbrace{|f(y_{n-k+1} - f(u_{n-k+1})|}_{L|y_{n-k+1} - u_{n-k+1}| = L|e_{n-k+1}|}\right) + h|\tau_n|.$$

How about bounding these by the largest error?

$$E_n := \max_{0 \le j \le n} |e_j|$$

This gives

$$\begin{aligned} |e_{n+1}| &\le \left[1 + h\underbrace{(|p_0| + \cdots + |p_{k-1}|)L}_{\text{indep. of } h}\right] E_n \\ &\le E_{n+1} \quad \text{(to see this, consider two cases:)} \end{aligned}$$

If $|e_{n+1}| \le E_n$, then $E_{n+1} = E_n \le RHS$ (this case is not very likely). On the other hand, if $|e_{n+1}| > E_{n+1} \le RHS$.

Notice this is just like Euler's method, where

$$E_n \le \frac{e^{pLnh} - 1}{hL}\left(h\tau - E_k\right)$$
$$p = |p_0| + \cdots + |p_{k-1}|$$

So as $h \to 0$, we have $E_n \to 0$ if:
(1) $\tau = O(h^k)$ which we already know, and
(2) $\frac{1}{h}E_{k-1} \to 0$, $\frac{1}{h}(u_0 - y_0) \to 0$, $\ldots$, $\frac{1}{h}(u_{k-1} - y_{k-1}) \to 0$. We compute these guys via Taylor expansion or a Runge-Kutta method with fewer steps. Notice that $\frac{1}{h} \to +\infty$, so the factor $(u_j - y_j)$ must go down faster to offset this.

4 Getting Started (Programatically)

For $n = 0 \to n = 1$, we have to use a 1-step method, say Euler-predictor:

$$\begin{aligned} v_n &= u_0 + hf(t_0, u_0) \quad \text{(predictor)} \\ u_1 &= u_0 + hf(t_1, v_1) \quad \text{(implicit Euler, corrector)} \end{aligned}$$

The problem has $O(h_0^2)$ error.
Eventually, we want $O(h^k)$ error.

$$h_0^2 = h^k \implies h_0 = T^{1-(k/2)}h^{k/2} << h$$

We have options. We can keep the time-step the same and increase the order. Alternatively, with a low-order method, we can double the time-step without doing any harm (this is a small win); hence we can increase the order and time-step simultaneously. For our purposes, we can just keep the step-size the same:

$$\begin{aligned} v_2 &= u_1 + h_0\left[\frac{3}{2}f_1 - \frac{1}{2}f_0\right] \\ u_2 &= u_1 + h\left[\frac{1}{2}f(t_2, v_2) + \frac{1}{2}f_1\right] \\ f_2 &= f(t_2, u_2) \\ &\vdots \\ f_{k-1} &= \cdots \end{aligned}$$

When we evaluate $f(t_1, v_1)$ for u_1, we need to compute and store $f_1 = f(t_1, u_1)$ as the predictor. We predict, evaluate, correct, then evaluate (PECE).
We use a 2-step method for the predictor v_2, so we use 1-step for u_2. Our time-steps are accurate to order h^{k+1} accuracy. The idea is that the process of getting started (with small steps), then gradually increasing our step-size, trying our steady increase θ from earlier until at some point, we want to check how much distance we have yet to cover and how much time we have to compute, then to re-define h so that we march along.
Essentially we might as well be coding a multi-step methods, but Strain doesn't want a riot!

5 Stiff Equations

In stiff equations, like weather, we have wind and chemistry that happens much faster than wind, and even smaller contributions such as fires.
We have a stiff equation in that if we follow the many events over so much detail over geological times, we can't use anything like multistep methods. It turns out, Runge-Kutta methods generally don't work either. We need to use methods specifically designed for stiff equations!

$$\begin{aligned} y_1'(t) &= -y_1(t) \\ y_2'(t) &= -100y_2(t) \end{aligned}$$

Euler's method here won't be too bad for y_1; however, for this same time-step, Euler's method will get worse and worse for y_2. To see this,

$$\begin{aligned} u_{n+1} &= u_n - hu_n = (1-h)u_n \\ v_{n+1} &= v_n - 100hv_n = (1-100h)v_n = -v_n \end{aligned}$$

and we say $h \approx 0.02$. If we had a factor 10000 instead of 100, our timestep h has to 'chase' the smallest unit.
The **cure** for stiff equations is to use implicit methods!

$$\begin{aligned} u_{n+1} &= u_n - hu_{n+1} \\ v_{n+1} &= v_n - 100hv_{n+1} \\ (1+100h)v_{n+1} &= v_n \\ v_{n+1} &= \frac{1}{1+100h}v_n \end{aligned}$$

In general, we'll have to solve nonlinear equations (as opposed to the linear solution here).
The problem is now that we have to solve an implicit (i.e. Euler) equation, say:

$$u_{n+1} = u_n + hf(u_{n+1})$$

We solve this in one of three ways (the Holy Trinity): Bisection, Fixed Point, Newton. This is probably a system in that u_n has many components that we cannot simultaneously find all at once. Hence we consider the latter two.
Fixed point says:

$$\begin{aligned} v = g(v) &= v_n - 100hv \\ g'(v) &= -100h, \end{aligned}$$

and we have $|g'(v)| \leq \frac{1}{2}$ if $h \leq \frac{1}{200}$, which is exactly what we were trying to avoid in our first example. Hence we're left with Newton, and it better work!
Newton delivers us exactly:

$$v_{n+1} = \frac{1}{1+100h}v_n$$

Now, the problem is that for Newton, we need f', and if this is a system, then we need the Jacobian matrix $Df(u)$ (it's a fact, we need this, unless we cheat via differentiation coefficients or interpolation). Else, we can take a course on optimization to solve this without knowing the derivative.

Lecture ends here.

We'll talk more about Stiff equations this week, as they occur a lot in real life. Next week we'll get started with Linear Algebra.

Math 128A, Summer 2019
Lecture 23, Wednesday July 31, 2019

1 Stiff Equations

- Linear Stability
- Nonlinear Stability

Today we're going to look at a different problem to test:

$$\begin{aligned} y' &= \lambda(y - \phi(t)) + \phi'(t) \\ (y-\phi)' &= \lambda(y-\phi) \\ y - \phi &= e^{\lambda t}[y(0) - \phi(0)] \\ y(t) &= \phi(t) + \underbrace{e^{\lambda t}[y(0) - \phi(0)]}_{\text{transient}} \end{aligned}$$

where $\lambda \in \mathbb{C}$ is the dampening constant if it's negative ($\text{Re}\lambda << 0$), and $\phi(t)$ describes something that oscillates smoothly (like a massage chair), and λ describes how we want to get to that smooth oscillation. We insert a forcing term $+\phi'(t)$ into the first line so this is solvable.
If we happen to start exactly at the correct spot, $\phi(0)$, then $y(0) = \phi(0)$.

$$e^{\lambda t} = e^{\alpha t}(\cos(\beta t) + i\sin(\beta t)), \qquad \text{when } \lambda = \alpha + i\beta$$

(Alternatively, we see this in an old car as we go over a bump; there is a number of oscillations before our shock absorbers stabilize the bumpiness.)
Last time, we observed that Euler's method does not work very well for this situation. To see this explicitly, consider:

$$\begin{aligned} u_{n+1} &= u_n + h\left[\lambda(u_n - \phi_n) + \phi_n'\right] \\ &= \underbrace{(1 + h\lambda)u_n}_{} + hg_n \end{aligned}$$

This is the scary part because if $|1 + h\lambda| > 1$, then u_{n+1} will oscillate instead of decaying. Alternatively, if $|\text{Re}(\lambda)| > 0$, then this will not blow up exponentially. This method still converges if $h \to 0$, but the issue is that if $\lambda << 0$ (extremely negative), then small h does too much work to follow the smooth solution $y(t) \approx \phi(t)$.

Remark: This restriction $|1 + h\lambda| < 1$ will be a very severe restriction, relative to accuracy (it is more than we need for accuracy; it is a very expensive hypothesis).

In the last lecture (22), we observed that **implicit euler** makes this problem go away, but it is only first-order accurate (and may be way too expensive as well). Additionally, in the last lecture, we found that we must solve by Newton's method because fixed point iteration will not converge.

Now, we want to generalize so that we can consider other methods (and not only Euler). One way that delivers 99% of the story is **linear stability theory**.

2 Linear Stability

The idea is we check the method on

$$y' := \lambda y, \qquad \text{Re}(\lambda) \leq 0$$

to verify that

$$|u_{n+1}| \leq |u_n|.$$

The least we should ask for is that the numerical method should decay, because it is generally not helpful to model something by an explosion.
Recall:

$$\begin{aligned}
\text{Euler: } u_{n+1} &= \overbrace{(1+h\lambda)}^{\text{Re}(z)} u_n \\
\text{Implicit Euler: } u_{n+1} &= u_n + h\lambda \underline{u_{n+1}} \\
(1-h\lambda)u_{n+1} &= u_n \\
u_{n+1} &= \underbrace{\frac{1}{1-h\lambda}}_{\text{Re}(z)} u_n
\end{aligned}$$

We consider Trapezoidal Rule and Midpoint Rule to help with Implicit Euler (for u_{n+1}), where if we apply them to a linear problem, we get the same result:

$$\begin{aligned}
\text{TR: } u_{n+1} &= u_n + \frac{h}{2}\left[f(u_n) + f(u_{n+1})\right] = u_n + \frac{h}{2}\lambda(u_n + u_{n+1}) \\
\text{Midpoint: } u_{n+1} &= u_n + hf\left(\frac{u_n + u_{n+1}}{2}\right) \\
\left(1 - \frac{h}{2}\lambda\right) u_{n+1} &= \left(1 + \frac{h\lambda}{2}\right) u_n \\
\implies u_{n+1} &= \underbrace{\left[\frac{1 + h\lambda/2}{1 - h\lambda/2}\right]}_{\text{Re}(z)} u_n
\end{aligned}$$

Essentially, we can write

$$u_{n+1} = \text{Re}(z) \cdot u_n,$$

where we multiply by the Real part of some $z := h\lambda$. We call $\text{Re}(z)$ the "stability function". Notice Strain writes $R(z)$ instead of $\text{Re}(z)$, but indeed this is simply taking the real part.

Definition: Region of Absolute Stability (RAS) -

A numerical method for $y' = f(y)$ has a Region of absolute stability wherever:

$$\{z : |\text{Re}(z)| \leq 1\}$$

For example, in **explicit Euler**, $\text{Re}(z) := 1 + z$, and the region of absolute stability is **within** a unit circle in the complex plane centered around

$(-1, 0)$. What this tells us is that if we solve a problem for some given λ and associated angle in the complex plane polar coordinates, the scaling factor h must be small enough to land in the circle.
We already have other methods which are much better, say for instance implicit euler, where

$$\text{Re}(z) = \frac{1}{1-z}, \qquad |\text{Re}(z)| \leq 1, \qquad |1-z| \geq 1,$$

and the region for absolute stability is **outside** a unit circle centered about $(1, 0)$.
And in the case for Midpoint or Trapezoidal rule, we have:

$$|\text{Re}(z)| = \left|\frac{1+z/2}{1-z/2}\right| \leq 1, \qquad |2+z| \leq |2-z|,$$

To see that the region of absolute stability is all of the left complex plane, let $z := x + yi$ per usual:

$$\begin{aligned} |2+z|^2 &\leq |2-z|^2 \\ (2+x)^2 + y^2 &\leq (2-x)^2 + y^2 \\ 4 + 4x + x^2 + y^2 &\leq 4 - 4x + x^2 + y^2 \implies x \leq 0 \end{aligned}$$

It seems this is the best possible situation (but it's not, because setting $z := iy$ makes the solution oscillate rapidly, and $z := -\infty$). Although this seems to give the perfect region for absolute stability, it has its problems.

Definition: A-stable -

A numerical method is said to be **A-stable** if the Region of Absolute Stability (RAS) includes the **entire left half of the complex plane**.

The reason for this definition is that A-stable methods ought to be OK (good) for stiff problems. It turns out there's a sub-class that performs even better (L-stability).

Definition: L-stable -

A numerical method is **L-stable** if it is A-stable with the additional condition:

$$\lim_{|z|\to\infty} \text{Re}(z) = 0.$$

Theorems: that there is no A-stable multi-step method with $k \geq 2$. Additionally, no explicit Runge-Kutta method is A-stable.
We prove the second. We look at Runge-Kutta methods and write:

$$\begin{aligned} k_i &= f\left(u_n + h\sum a_{i,j}k_j\right) \\ &= \lambda\left(u_n + h\sum a_{i,j}k_j\right) \\ \implies k_i - h\lambda\sum a_{i,j}k_j &= \lambda u_n \\ u_{n+1} &= u_n + \underbrace{h}\sum b_i k_i \end{aligned}$$

What we are doing here is multiplying for each i, k is a vector:

$$\begin{aligned}
[I - h\lambda A]\, k &= \lambda u_n e, \qquad e := [1; 1; \cdots; 1] \\
k &= [I - zA]^{-1} \lambda u_n e \\
&= \lambda u_n [I - zA]^{-1} e \\
u_{n+1} &= u_n + hb^T k \\
&= u_n + \underbrace{h\lambda}_{z} u_n b^T [I - zA]^{-1} e \\
&= \left[1 = zb^T [I - zA]^{-1} e\right] u_n,
\end{aligned}$$

so we proved that the region of absolute stability for a Runge Kutta method can be read off from the Butcher array:

$$R(z) = 1 + zb^T [I - zA]^{-1} e,$$

where for example in Trapezoidal Rule,

$$\begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

and hence:

$$u_{n+1} = u_n + \frac{h}{2}\left[\underbrace{f(u_n)}_{k_1} + \underbrace{f(u_{n+1})}_{k_2}\right]$$

and hence inverting the matrix $[I - zA]$, we have:

$$\begin{aligned}
[I - zA]^{-1} e &= \begin{bmatrix} 1 & 0 \\ \frac{-z}{2} & 1 - \frac{z}{2} \end{bmatrix}^{-1} e \\
&= \frac{1}{1 - z/2} \begin{bmatrix} 1 - \frac{z}{2} & 0 \\ \frac{z}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \\
&= \frac{1}{1 - z/2} \begin{bmatrix} 1 - z/2 \\ 1 + z/2 \end{bmatrix},
\end{aligned}$$

and hence:

$$\begin{aligned}
R(z) &= 1 + z\frac{1}{1 - z/2}\left[\frac{1}{2}\left(1 - \frac{z}{2}\right) + \frac{1}{2}\left(1 = \frac{z}{2}\right)\right] \\
&= \frac{1 - z/2 + z}{1 - z/2} = \frac{1 + z/2}{1 - z/2} \qquad ,
\end{aligned}$$

which is the correct expression as we derived earlier today.

Break time. After the break we'll apply this to the geometric series (taylor expansion).

Now they taylor series of geometric series on $(I - zA)$ gives:

$$(I - zA)^{-1} = I + zA + z^2A^2 + z^3A^3 + \cdots + z^pA^p,$$

if A is nilpotent and $A^{p+1} = 0$. For example, in an **explicit** Runge-Kutta method, our diagonal values of the matrix A are all zero, so successive powers cause the zeros to 'march down'.

Remark: For any explicit Runge Kutta method, the matrix A is strictly lower triangular, so $A^s = 0$ and $R(z)$ is a polynomial, which can be seen as $\lim_{|z|\to\infty} R(z) = +\infty$. Hence we proved that any **explicit** Runge Kutta method cannot be A-stable.

2.1 Trying for L-stability

Recall that the Trapezoidal rule is second-order accurate, so we want something slightly better. Now let's consider making a a **second-order implicit Runge-Kutta method** which is L-stable. We try **deferred correction** on implicit euler, and maybe it will be L-stable or A-stable.

Try $p = 2$. We first take a step of implicit euler:

$$\begin{aligned} u_{n+1} &= u_n + hf(u_{n+1}) \\ u_{n+2} &= u_{n+1} + hf(u_{n+2}) \\ u(t) &= \cdots u_n, u_{n+1}, u_{n+2} \end{aligned}$$

Next we look at the errors. $e_n := 0$.

$$\begin{aligned} e_{n+1} &= e_n + h\left[f(u_{n+1} + e_{n+1}) - u'(t_{n+1})\right] \\ e_{n+2} &= e_{n+1} + h\left[f(u_{n+2} + e_{n+2}) - u'(t_{n+2})\right] \\ u^1_{n+1} &= u_{n+1} + e_{n+1} \\ u^1_{n+2} &= u_{n+2} + e_{n+2} \\ &= u_n + 2h\left[b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4\right] \end{aligned}$$

Our plan here is to find $|R(z)| \leq 1$. We don't feel like doing the tedious algebra, so we fix $h := \lambda$ and vary $\lambda = z$ over some box in the left side of the complex plane. Because we don't know the values of coefficents b_i, we just use our above equation

$$u^1_{n+2} = u_{n+2} + e_{n+2}$$

and assert

$$|u_{n+1}| \leq 1 \implies z \in RAS$$

To do this, we write:

$$\begin{aligned} u_{n+1} &= \frac{1}{1 - h\lambda} u_n \\ u_{n+2} &= \frac{1}{(1 - h\lambda)^2} u_n \\ &\vdots \end{aligned}$$

Notice that our invented method is not a multistep method, as we go from u_n to u_{n+2}, moving two steps at a time.
Alternatively, we can find $\{z : |R(z)| = 1\}$ which is called the 'boundary locus technique'.

3 Nonlinear Stability

Nonlinear stability is a lot simpler than linear stability. In linear stability, we said $y' = \lambda y$, and $|\lambda| < 0$. For assertions or conditions on nonlinear stability, we cannot say that the solution goes to 0 because the exact solution may not be zero. The real question if the exact solutions get closer together? We look at two different trajectories with $1 \leq j \leq p$:

$$u_j'(t) = f_j(u(t))$$
$$v_j'(t) = f_j(v(t)),$$

where we don't worry about t because we can make these autonomous. Looking at the distance between these, we have:

$$||u(t) - v(t)|| = \sqrt{\sum_j (u_j(t) - v_j(t))^2},$$

where we want the distance to decrease as t increases. We use the euclidean norm because it is simple when we differentiate. We want to assert:

$$\begin{aligned}\frac{d}{dt}||u(t) - v(t)||^2 &= \frac{d}{dt}\sum_j [u_j(t) - v_j(t)]^2 \\ &= \sum_j 2\left(u_j(t) - v_j(t)\right)\left(u_j'(t) - v_j'(t)\right) \\ &= \sum_j 2\left(u_j - v_j\right)\left(f_j(u) - f_j(v)\right) \leq 0,\end{aligned}$$

where we want to assert this is true for all pairwise elements (and all t). This is equivalent to saying:

$$\frac{d}{dt}||u(t) - v(t)||^2 = 2(u - v)^T\left[f(u) - f(v)\right]$$

This all suggests a definition:

Definition: Dissipative -

We say f is dissipative if

$$\left(f(u) - f(v)\right)^T(u - v) \leq 0$$

Many functions are dissipative, such as if we have a positive-definite or negative-definite function.
Our above calculations above prove a theorem:

Theorem 3.1. If function f is dissipative and $u' = f(u), v' = f(v)$, then

$$||u(t) - v(t)||$$

is decreasing.

Our goal is to design a method for which the distance between two different numerical solutions get closer to gether whenever f is dissipative. We call this "contractive" (except we don't have the factor of $\frac{1}{2}$). That is,

$$||u_{n+1} - v_{n+1}|| \leq ||u_n - v_n|| \qquad \text{(contractive)}$$

For example, let us take implicit euler. We know this is a nice L-stable method.

$$u_{n+1} = u_n + hf(u_{n+1})$$
$$v_{n+1} = v_n + hf(v_{n+1}),$$

and substracting yields

$$v_{n+1} - v_{n+1} = u_n - v_n + h(f(u_{n+1}) - f(v_{n+1}))$$

If we dot both sides of this equation, we have:

$$||u_{n+1} - v_{n+1}||^2 \leq (u_{n+1})^T(u_n - v_n) + \underbrace{h(u_{n+1} - v_{n+1})^T(f(u_{n+1}) - f(v_{n_1}))}_{\leq 0 \text{ if } f \text{ dissipative}}$$
$$\leq \underbrace{||u_{n+1} - v_{n+1}||}\,||u_n - v_n||$$
$$\implies ||u_{n+1} - v_{n+1}|| \leq ||u_n - v_n||$$

By the Cauchy Schwarz inequality, the dot product of two vectors is less than the product of their lengths, to get the last inequality. We call this B-stable, which brings us to the corresponding definition:

Definition: B-stable -

We say a numerical method is B-stable if

$$||u_{n+1} - v_{n+1}|| \leq ||u_n - v_n|| \qquad (\text{"contractive"})$$

whenever f is dissipative.

Dissipative functions f guarantee a specific type of stability in differential equations.

Lecture ends here.

4 Key Results

- Region of Absolute Stability
- Definitions: RAS, A-stable,
- Implicit Euler is first-order accurate
- Trapezoidal/Midpoint Rules are second-order accurate
- Additional conditions can give A-stability or L-stability

5 Office Hours

$$\begin{aligned} y_{n+1} &= y_n + \sum p_j \underline{f_{n-j}} + h\tau_n \\ &= y_n + \int_{t_n}^{t_{n+1}} [y'(s) - p(s)] + \underline{p(s)}\ ds \\ h\tau_n &= \int_{t_n}^{t_{n+1}} \underbrace{y'(s) - p(s)}\ ds \end{aligned}$$

where this error is:

$$\frac{y^{(k+2)}}{(k+1)!}\omega(s).$$

The reason for the mismatch between the derivative order and the denominator factorial is that we are interpolating the derivative!

Math 128A, Summer 2019

Lecture 24, Today's Date 8/1/2019

1 Review

Recall from yesterday that we have a definition that f is disappative means that

$$(f(u) - f(v))^T (u - v) \leq 0.$$

Additionally, recall that we have a theorem that if f is dissipative and $u'(t) = f(u)$ and $v'(t) = f(v)$, then

$$||u(t) - v(t)|| \qquad \text{decreasing}$$

This is a desirable because this means that our method forgets our initial conditions. This is good if we want to 'destroy' evidence, assuming we know where we want to be. To put these neatly, we include the following from lecture 23 yesterday.

Definition: Dissipative -

We say f is dissipative if

$$(f(u) - f(v))^T (u - v) \leq 0$$

Theorem 1.1. If function f is dissipative and $u' = f(u), v' = f(v)$, then

$$||u(t) - v(t)||$$

is decreasing.

Definition: B-stable -

We say a numerical method is B-stable if

$$||u_{n+1} - v_{n+1}|| \leq ||u_n - v_n|| \qquad (\text{"contractive"})$$

whenever f is dissipative.

2 Nonlinear Stability Theory

We claim the following.

Theorem 2.1. B-stable implies A-stable.

Proof. Notice that B-stable is a nonlinear property, whereas A-stable is a linear property. Recall that if a method is A-stable, we have $y' = \lambda y, \mathrm{Re}\lambda \leq 0, |u_{n+1}| \leq |u_n|$.

$$\begin{aligned} y &= u + iv \\ y' &= u' + iv' \\ &= (\alpha + i\beta)(u + iv) \\ &= \alpha u - \beta v + i(\alpha v + \beta u) \\ \begin{bmatrix} u \\ v \end{bmatrix}' &= \underbrace{\begin{bmatrix} \alpha & -\beta \\ \alpha & \beta \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}}_{f(u,v)} \end{aligned}$$

Applying the fact that these are linear, with $\alpha \leq 0$, consider:

$$\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \alpha u - \beta v \\ \beta u + \alpha v \end{bmatrix} = \alpha u^2 - \beta uv + \beta uv + \alpha u^2 \leq 0,$$

and we are done.

□

3 Algebraic Stability

Now we want to check for B-stability for Runge-Kutta methods by **algebra** only.

$$k_i = f(\underbrace{u_n + h \sum a_{i,j} k_j}_{=:q_i})$$

Hence

$$\begin{aligned} u_{n+1} &= u_n + h \sum b_i f(q_i) \\ &= u_n + h \sum b_i k_i \end{aligned}$$

We start somewhere else,

$$\begin{aligned} k_i^* &= f(u_n^* + h \sum a_{i,j} k_j^*) \\ u_{n+1}^* &= u_n^* + h \sum b_i k_i^*, \end{aligned}$$

which gives a different solution, u^*.
Taking the difference between these two, we have:

$$\begin{aligned} \underbrace{u_{n+1} - u_{n+1}^*}_{=:\Delta u_{n+1}} &= u_n - u_n^* + h \sum b_i (k_i - k_i^*) \\ \Delta u_{n+1} &= \Delta u_n + h \sum b_i \Delta k_i \\ \Delta k_i &= f(q_i) - f(q_i^*) \end{aligned}$$

The reason for inserting these q_i is that if we dot the difference between the q's and multiply by k_i, then this will be ≤ 0 if f is dissipative. That is, if f is disappative,

$$\Delta q_i^T \Delta k_i \leq 0$$
$$(q_i - q_i^*)^T (f(q_i) - f(q_i^*)) \leq 0$$

Taking norms squared, we have:

$$||\Delta u_{n+1}||^2 = ||\Delta u_n||^2 + 2h(\Delta u_n)^T \sum b_i \Delta k_i + h^2 \left(\sum_i b_i \Delta k_i\right)^T \left(\sum_j b_j \Delta k_j\right)$$

To help, consider:

$$q_i = u_n + h \sum a_{i,j} k_j$$
$$\Delta q_i = \Delta u_n + h \sum a_{i,j} \Delta k_j$$
$$\Delta u_n = \Delta q_i - h \sum a_{i,j} \Delta k_i.$$

Now we should write down the important part with the $2h$ in front:

$$\Delta u_n^T \Delta k_i = \Delta q_i^T \Delta k_i - h \sum a_{i,j} \Delta k_j^T \Delta k_i$$

And we can now rewrite the whole thing with what we figured out:

$$||\Delta u_{n+1}||^2 = ||\Delta u_n||^2 + 2h \sum b_i \left(\Delta q_i^T \Delta k_i - h \sum_j a_{i,j} \Delta k_i^T \Delta k_j\right) + h^2 \sum b_i \Delta k_i^T \sum b_j \Delta k_j$$

Because the method is B-stable, f is dissipative, and thus $\Delta q_i^T \Delta k_i \leq 0$, which gives us:

$$||\Delta u_{n+1}||^2 \leq ||\Delta u_n||^2 + h^2 \sum_i \sum_j \left[(b_i b_j - 2b_i a_{i,j}) \Delta k_i^T \Delta k_j\right]$$

Notice that we can swap i and j, because the matrix $\Delta k_i^T \Delta k_j$ is symmetric (via effective dot product):

$$\sum_i \sum_j b_i a_{ij} \Delta k_i^T \Delta k_j = \sum_i \sum_j b_j a_{ji} \Delta k_i^T \Delta k_j$$

and now because we have two, we take one of each representation.

$$||\Delta u_{n+1}||^2 \leq ||\Delta u_n||^2 + h^2 \sum_i \sum_j \overbrace{(b_i b_j - b_i a_{ij} - b_{ji} a_{ji})}^{M_{ij}} \Delta k_i^T \Delta k_j$$

We conclude that if $b_i \geq 0$ and $x^T M x \leq 0$ for all vectors x, then the method is B-stable. We call this **algebraic stability**.
How do we check $x^T M x \leq 0$ for all x? Because M is a symmetric matrix, M has a basis of eigenvectors, say $e_1, \ldots, e_s$.
This gives:

$$e_j^T M e_j = e_j^T \lambda_j e_j,$$

if $||e_j|| = 1$. Hence this is equivalent to checking $\lambda_j \leq 0$ for $\forall j$. This is the condition for M to be negative semi-definite.

3.1 Example: Trying Implicit Euler

Recall that the Butcher Array Implicit Euler is simply $\begin{bmatrix}1\\1\end{bmatrix}$, where the lower value is $1 \geq 0$. Hence

$$M = 1 \cdot 1 - 1 - 1 = -1 \leq 0,$$

and hence Implicit Euler is indeed B-stable (which we already knew).

3.2 Example: Trying Trapezoidal Rule

$$\begin{aligned} k_1 &= f(u_n) \\ k_2 &= f(u_{n+1}) = f\left(u_n + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2)\right)\right) \\ u_{n+1} &= u_n + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right), \end{aligned}$$

which gives us the Butcher array:

$$\begin{bmatrix} 0 & 0 \\ 1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

where the bottom row is all ≥ 0. Now we need to evaluate the matrix M:

$$\begin{aligned} M &= \begin{bmatrix}1/2\\1/2\end{bmatrix}\begin{bmatrix}1/2 & 1/2\end{bmatrix} - \begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix}\begin{bmatrix}0 & 0\\1/2 & 1/2\end{bmatrix} - \begin{bmatrix}0 & 1/2\\0 & 1/2\end{bmatrix}\begin{bmatrix}1/2 & 0\\0 & 1/2\end{bmatrix} \\ &= \begin{bmatrix}-1/4 & 1/4\\1/4 & 1/4\end{bmatrix} - \begin{bmatrix}0 & 0\\1/4 & 1/4\end{bmatrix} - \begin{bmatrix}0 & 1/4\\0 & 1/4\end{bmatrix} = \begin{bmatrix}1/4 & 0\\0 & -1/4\end{bmatrix} \end{aligned}$$

We aren't sure of this result (that Trapezoidal Rule is not B-stable), so we check the Midpoint rule. If this is the case, Trapezoidal Rule is A-stable but not B-stable.

3.3 Midpoint Rule

$$\begin{aligned} u_{n+1} &= u_n + hf\left(\frac{u_n + u_{n+1}}{2}\right) \\ k_1 &= f\left(\frac{u_n}{2} + \frac{1}{2}\left(u_n + hk_1\right)\right) \\ &= f\left(u_n + \frac{1}{2}hk_1\right) \end{aligned}$$

which gives the butcher array

$$\begin{bmatrix}1/2\\1\end{bmatrix}$$

and we write:

$$M = 1 - \frac{1}{2} - \frac{1}{2} = 0 \leq 0,$$

so the Midpoint rule is B-stable but Trapezoidal rule is **not** B-stable (technically we only failed to prove it is B-stable, and not exactly proved it isn't).

4 Hammer-Hollingsworth

This is a fourth-order, two stage implicit Runge-Kutta method which is B-stable (4-order 2-stage IRK).
We write this as

$$\int_{t_n}^{t_{n+1}} y'(s)\ ds = w_1 y'(s_1) + w_2 y'(s_2) = y_{n+1} - y_n + \underbrace{O(h^5)}_{h\tau_n}$$

To get the RHS of the above, let's say we use 2-point Gaussian integration, which we recall from before takes on points at $\pm\frac{1}{\sqrt{3}}$ with weights that add to 2. Let s_1, s_2 be the Gaussian Quadrature points:

$$s_1 = t_n + \frac{h}{2} + \frac{h}{2}\left(\frac{-1}{\sqrt{3}}\right)$$
$$s_2 = t_n + \frac{h}{2} + \frac{h}{2}\left(\frac{+1}{\sqrt{3}}\right)$$

So we take $w_1 = w_2 := \frac{h}{2}$.
How do we find the stages?

$$k_1 \approx y'(s_1)$$

This means that we evaluate:

$$\underbrace{f(u_n + h\sum_{j=1}^{2} a_{ij}k_j)}_{y(s_1)},$$

where

$$y(s_1) = y_n + \int_{t_n}^{s_1} y'(s)\ dt$$

The stages
We observed that there is a chance of getting fourth order accuracy if we put the stages precisely at the Gauss points. This tells us what the c's are, but it doesn't give us the a's that we need. The row sums of the Butcher array are:

$$c_1 = \frac{1}{2} - \frac{1}{2\sqrt{3}}$$
$$c_2 = \frac{1}{2} + \frac{1}{2\sqrt{3}}$$

and we have some Butcher array:

$$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

for some matrix up top.

So, we want to do the integral:

$$\int_{t_n}^{s_i} y'(s)\ ds = h \sum \underbrace{a_{ij}}\ y'(s_j)$$

There aren't many ways to get these a_{ij}, so we say to make it exact for the function $f(x) := 1$, then for the function $f(x) = s$:

$$\begin{aligned}
&\int_{t_n}^{s_i} 1\ ds = h \sum_j a_{1j} 1 \\
\implies\ & s_i - t_n = h\left(a_{i,1} + a_{i,2}\right) \\
&\int_{t_n}^{s_i} s\ ds = h \sum_j a_{1j} s_j \\
\implies\ & \frac{1}{2} s^2 |_{t_n}^{s_i} = h\left(a_{i1} s_1 + a_{i2} s_2\right) \\
& \frac{1}{2} s_i^2 - \frac{1}{2} t_n^2 = h\left(a_{i1} s_1 + a_{i2} s_2\right),
\end{aligned}$$

and we skip the algebra and write the resulting Butcher array:

$$\begin{bmatrix}
\left(\frac{1}{2} - \frac{1}{2\sqrt{3}}\right) | & \frac{1}{4} & \left(\frac{1}{4} - \frac{1}{2\sqrt{3}}\right) \\
\left(\frac{1}{2} + \frac{1}{2\sqrt{3}}\right) | & \left(\frac{1}{4} + \frac{1}{2\sqrt{3}}\right) & \frac{1}{4} \\
& \frac{1}{2} & \frac{1}{2}
\end{bmatrix}$$

We call this **Hammer-Hollingsworth** '2-stage Gauss'. To check if this is B-stable, we do the easy part first, which is to check the bs at the bottom are ≥ 0, which is true.
Now,

$$\underbrace{\begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix}}_{bb^T} - \underbrace{\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}}_{B=\mathrm{diag}(b)} \underbrace{\begin{bmatrix} \left(\frac{1}{4}\right) & \left(\frac{1}{4} - \frac{1}{2\sqrt{3}}\right) \\ \left(\frac{1}{4} + \frac{1}{2\sqrt{3}}\right) & \frac{1}{4} \end{bmatrix}}_{A} - \underbrace{\begin{bmatrix} \frac{1}{4} & \left(\frac{1}{4} + \frac{1}{2\sqrt{3}}\right) \\ \left(\frac{1}{4} - \frac{1}{2\sqrt{3}}\right) & \frac{1}{4} \end{bmatrix}}_{A^T} \underbrace{\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}}_{B}$$

$$= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \leq 0,$$

so Hammer-Hollingsworth is indeed B-stable!

5

Now suppose $g : \mathbb{R}^m \to \mathbb{R}$, and we want to minimize $g(x)$. To do this, first notice that our result is a single result which makes this very simple. To not brute-force or perform random searches, we want from calculus:

$$\begin{aligned}
x'(t) &= -\nabla g(x) \\
&= f(y)
\end{aligned}$$

Essentially, this can work like (stochastic) gradient descent. Our question is whether f is dissipative. To see this,

$$(u - v)^T (f(u) - f(v)) = (u - v)^T (-\nabla g(u) + \nabla g(v)).$$

If we have a one-dimentional problem, we have:

$$\begin{aligned}(g'(v) - g'(u))(u - v) &= g''(\xi)(v - u)(u - v) \\ &= -g''(\xi)(u - v)^2 \\ &\leq 0,\end{aligned}$$

if $g''(\xi) \geq 0$. This is what it means for f to be dissipative (this is why dissipative systems come up a lot). Unfortunately, in multi-dimensions, there is **no reason** for the mean-value theorem to apply, as the mean value may not lie on the exact straight line between two points.
However, we still have:

$$\begin{aligned}u - v &= v + t(u - v)|_{t=0}^{t=1} \quad \text{(just verify this is true)} \\ &= \int_0^1 \frac{d}{dt}(v + t(u - v))\end{aligned}$$

Hence

$$\begin{aligned}(u - v)^T (f(u) - f(v)) &= \int_0^1 \frac{d}{dt}(v + t(u - v))\ dt^T (f(u) - f(v)) \\ &= \int_0^1 \frac{d}{dt}[v + t(u - v)]^T (f(u) - f(v))\ dt\end{aligned}$$

To solve this integral, we'll pick up on Monday!

Lecture ends here.

Math 128A, Summer 2019
Lecture 25, 8/5/2019

CLASS ANNOUNCEMENTS: Linear Algebra (7 lectures):
- Direct methods for solving linear systems, $Ax = b$.
- Special Types of Matrices: (1) Diagonally Dominant, (2) Positive Definite
- Linear Algebra Under Floating Point Arithmetic (proving that it can work)
- Overcoming Limitations (Iterative Improvement) of Solution to Linear Systems

1 Review

We define a matrix $A_{n\times n}$ as a box of numbers, because we commit the sin of getting our hands dirty, away from pure mathematics. The box of numbers is with respect to some pair of bases, so sometimes we treat a matrix simply as a box of numbers without reference to a particular basis and linear operator.

The space of these matrices is usually called $\mathbb{R}^{n\times n} = \mathbb{R}^{n^2} = L(\mathbb{R}^n, \mathbb{R}^n)$ (linear operator).

We define a vector as a column vector (matrix). We write **Matrix-vector multiplication** as Ax (left-multiplication). We take the standard basis:

$$e_1 = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \quad e_2 = \begin{bmatrix}0\\1\\.\\0\\0\end{bmatrix}, \quad \cdots, \quad e_n = \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$$

We write vector multiplication as a dot product, taking the transpose:

$$e_i^T e_j = \begin{bmatrix}\cdot & \cdot & \cdots & \cdot\end{bmatrix} \begin{bmatrix}\cdot\\\cdot\\\vdots\\\cdot\end{bmatrix} = \delta_{ij} = \begin{cases}1, & i = j\\0, & i \neq j\end{cases}$$

, where δ_{ij} is the Kronecker delta.
Alternatively, we can take the other transpose:

$$e_i e_j^T$$

which gives a matrix full of 0's except for 1 at the i, jth entry.
We can write the identity matrix in terms of this basis (for $n \times n$ matrices):

$$I = \begin{bmatrix}1 & 0 & \cdots & 0\\0 & 1 & \cdots & 0\\\vdots & \vdots & \ddots & \vdots\\0 & 0 & \cdots & 1\end{bmatrix} = e_1e_1^T + e_2e_2^T + \cdots + e_ne_n^T = \sum_{i=1}^{n} e_ie_i^T = \sum_{j=1}^{n} e_je_j^T$$

So we can write any A as

$$A = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \left(e_i e_j^T \right)$$

$$A = AI = A \sum_{j=1}^{n} e_j e_j^T = \sum_{j=1}^{n} (Ae_j) \underbrace{e_j^T}_{},$$

putting this in column j of A, and putting the following in row i of A

$$IA = \sum_{i=1}^{n} \overbrace{e_i} \left(e_i^T A \right)$$

We can also write, for **extracting matrix entries** if given some black box:

$$IAI = \sum_i \sum_j e_i \left(e_i^T A e_j \right) e_i^T = \sum \left(e_i^T A e_j \right) e_i e_j^T$$

and our convention is to write the scalar inside the parenthesis out to the front of the expression (and lose our structure and intuition).

Now we define matrix-vector multiplication. First we write:

$$X = Ix = \sum_{i=1}^{n} e_i \left(e_i^T x \right) = \sum_{i=1}^{n} x_i e_i, \qquad e_i := e_i^T x$$

and for matrix-vector multiplication,

$$\begin{aligned} (Ax)_i &= \sum_{j=1}^{n} a_{ij} x_j, \qquad x = \sum_i x_i e_i \\ &= \sum_{j=1}^{n} e_i^T A e_j e_j^T x, \end{aligned}$$

and equivalently,

$$\begin{aligned} Ax = AIx &= A \sum_i \left(e_i^T x \right) e_i \\ &= \sum_j \underbrace{\left(e_j^T x \right)}_{x_j} \underbrace{Ae_j}_{j \text{ col of } A} \end{aligned}$$

and we define $R(A)$ as the 'range' of A (or 'column space') to bring us to our first interpretation of matrix-vector multiplication. Strain's remark is that in linear algebra, we usually have two or four interpretations of each thing. Equivalently, we have our second interpretation:

$$Ax = IAx = \sum_{i=1}^{n} \overbrace{e_i}^{\text{put in row } i} \underbrace{e_i^T A}_{\text{row vec}} \underbrace{x}_{\text{col vec}}$$

$$I = \sum_{i=1}^{n} \underbrace{e_i}_{\text{col vec}} \underbrace{e_i^T}_{\text{row vec}}$$

Now we define Matrix-Matrix multiplication:

$$MA = (IM)(AI) = \sum e_i \underbrace{(e_i^T M)}_{\text{row}} \sum \underbrace{(Ae_j)}_{\text{col}} e_j^T$$

equivalently, we can place the identities differently:

$$MA = M \sum (Ae_j) e_j^T = \sum \underbrace{(MAe_j)}\underbrace{e_j^T}$$

and we say M hits each **column** of A, and e_j^T puts this into column j of the result. This means M does **row** operations on A.
If we right-multiply A by B now, we can expect that B does column operations on A. To see this, we write:

$$AB = \left(B^T A^T\right)^T = \sum_i e_i \left(e_i^T A\right) B.$$

We usually use left-multiplication, but there is always the symmetric viewpoint.
We can also write:

$$AB = \sum_j Ae_j e_j^T \sum_i e_i e_i^T B = \sum_{ij} Ae_j \underbrace{e_j^T e_i}_{\delta_{ij}} e_i^T B = \sum_i \underbrace{(Ae_i)}_{\text{col } i \text{ of } A} \underbrace{\left(e_i^T B\right)}_{\text{row } i \text{ of } B},$$

and we express AB as a sum of rank-1 $n \times n$ matrices.

2 Solving $Ax = b$

We already know how to do this via Row operations (row echelon forms). Say

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 15 \\ 15 \end{bmatrix}$$

We may write something like:

$$\begin{bmatrix} \underline{1} & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -21 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \\ -27 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & \underline{-2} & -6 \\ 0 & 0 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \\ 9 \end{bmatrix}$$

and we have a nice (easy) triangular system. We have: $\boxed{x_3 = 9}$, so $-3x_2 - 6x_3 = -9$ and thus $\boxed{x_2 = 1}$. Lastly, $x_1 + 2x_2 + 3x_3 = 6$, so we have $\boxed{x_3 = 1}$. The underlined main diagonal entries (pivots) must be nonzero for our system to have a (unique) solution.

Remark: Our algorithm (if possible) has to ensure nonzero pivots.

For an example of when this is not possible, if we have something like:

$$\begin{bmatrix} 0 & 1 & 4 \\ 0 & 2 & 5 \\ 0 & 3 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix},$$

then we cannot readily predict if we have a solution or if we have (infinitely) many solutions. It follows via linear algebra that here the kernel (null space) is 1-dimensional, so the solution space (range) is $(3-1)$-dimensional.
Strain: This seems like a good opportunity for a theorem.

Theorem 2.1. $\exists!$ solution x to $Ax = b$ if and only if:
(1) A is an intertible matrix, **independent** of choices of b, or
(2) $\det A \neq 0$, or
(3) no 0 pivots are encountered (with the correct sorting in Gaussian elimination, aka partial pivoting), or
(4) none of As eigenvalues is 0

(Aside) Strain: Applied math majors don't drink beer, they drink whiskey.

3 GEPP: Gaussian Elimination with Partial Pivoting

Now we say the real meaning behind **GEPP** (Gaussian Elimination with Partial Pivoting) is that if we take an equation $Ax = b$ and convert it into an upper-triangular form $Ux = \beta$, where U is upper-triangular with **nonzero entries on diagonal**. Recall that the entire first hour of lecture was spent on row operations. We say that there should be some operator M to enact row operations on A via MA to give $MA = U$. (We postpone discussion on pivoting until later, possibly tomorrow.) Our question is how do we find M that performs a specified row operation?
Strain (Aside): Suppose we are out drinking with a friend, and there's a bullet flying toward me, and I want to know what happens. I take my friend, place him in front of me and observe. We do this by inserting I:

$$MA = (MI)A$$

For example, to get from :

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \xmapsto{M_1} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -27 & -42 \end{bmatrix},$$

we take

$$M_1 := \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{bmatrix}$$

which we call **unit-lower-triangular**. Now we say

$$M_2 M_1 A = U$$

where U is upper-triangular and

$$M_2 := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 9 & 1 \end{bmatrix}.$$

$$\begin{aligned} M_2 M_1 A &= U \\ A &= LU \\ L &:= (M_1^{-1} M_2^{-1}) \end{aligned}$$

In summary, we find that $A = LU$ is actually simply just a matrix-factorization, with $L := \left(M_1^{-1} M_2^{-1}\right)$, which we call **lower-triangular reduction to upper-triangular form**.
In our previous example, we write:

$$M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -9 & 1 \end{bmatrix},$$

$$M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 0 & 1 \end{bmatrix}$$

Try:

$$M_1^{-1}M_2^{-1} = \begin{bmatrix}1 & 0 & 0\\ 4 & 1 & 0\\ 7 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & -9 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 4 & 1 & 0\\ 7 & -9 & 1\end{bmatrix},$$

and notice that this is **not** how matrix-matrix multiplication usually works. Here, we simply take items and place them into the final matrix. It turns out that L, which is apparently a relatively complicated object because we need to invert matrices, is actually a matrix of multipliers:

$$L = \begin{bmatrix}1 & 0 & \cdots & 0\\ m_{21} & \ddots & & 0\\ \vdots & & \ddots & 0\\ m_{n1} & \cdots & m_{n,m-1} & 0\end{bmatrix}$$

where m_{ij} is simply the row operations we perform for Gaussian elimination. For example, write:

$$A \mapsto M_1 A = \begin{bmatrix}a_1 1 & \cdots & a_{1n}\\ 0 & \ddots & \\ 0 & \cdots & \cdots\end{bmatrix}, \begin{bmatrix}a_{11} & c^T\\ d & A_{22}\end{bmatrix} \xrightarrow{e_i \frac{d_i}{a_{11}} c^T} \begin{bmatrix}a_{11} & c^T\\ \mathcal{M}(0) & A_{22} - \frac{d_i}{a_{11}}c^T\end{bmatrix},$$

where

$$M_1 := \begin{bmatrix}1 & \cdots & \cdots\\ \frac{-a_{21}}{a_{11}} & \ddots & \cdots\\ \vdots & \ddots & \cdots\\ \frac{-a_{n1}}{a_{11}} & \cdots & \cdots\end{bmatrix} = I - \frac{1}{a_{11}}\overbrace{(a_{:,1} - a_{11}e_1)}^{m_1} e_1^T,$$

where $e_1^T m_1 = 0$, where m_1 is the first column of A but setting the first entry (pivot) to 0. Notice that m_2 is the second column of **the result** after performing M_1 (to yield a different A).
Thus our theorem is essentially proven by a single step of recursion, where if A is invertible, then the smaller bottom-right parts are also invertible

Lecture ends here.

Next time we'll talk about **pivoting**. Strain jokes that usually once we learn pivoting, we end up trying to forget it quickly.

Math 128A, Summer 2019
Lecture 26, 8/6/2019

1 Pivoting

In real arithmetic, performing Gaussian elimination on something like

$$\begin{bmatrix} 1 & 2 \\ 3000 & 4 \end{bmatrix}$$

is perfectly fine. However, in computer arithmetic, subtracting a large number by a very small number is bad, because any error grows. Hence we want to perform **pivoting** to move rows and columns around via 'swaps'.
We want to do something like:

$$A = \begin{bmatrix} 1 & 2 \\ 3000 & 4 \end{bmatrix} \mapsto PA = \begin{bmatrix} 3000 & 4 \\ 1 & 2 \end{bmatrix},$$

where $P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Notice $P^{-1} = P$ and $P^2 = I$. Let's check if the general case for $P^{1,i}$ is a symmetric matrix:
To swap i, j rows,

$$\begin{aligned} P^{ij} &= I - e_i e_i^T - e_j e_j^T + e_i e_j^T + e_j e_i^T \\ &= I - (e_i - e_j)(e_i - e_j)^T \end{aligned}$$

Strain reminds us to keep the multipliers ≤ 1 to not lose information we may want to keep. To achieve this, consider Gaussian Elimination with Partial Pivoting (GEPP):

Definition: Partial Pivoting -

The simplest strategy, called **partial pioting**, is to select an element in the same column that is below the diagonal and **has the largest absolute value**. Specifically, we determine the smallest $p \geq k$ with:

$$|a_{pk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|$$

If $|a_{11}| = \max_{1 \leq i \leq n} |a_{i1}|$ then $m_{i1} \leq 1$ for all i.

We construct:

$$M_{n-1}P_{n-1} \cdots M_3 P_3 M_2 P_2 M_1 P_1 A = U = \begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ 0 & \cdots & u_{nn} \end{bmatrix}$$

where M_i are unit lower-triangular, and P_i performs row swaps, so U is nonsingular if and only if A is nonsingular.
We claim that if all these P above are absent, then

$$M_{n-1} \cdots M_3 M_2 M_1 = \begin{bmatrix} 1 & & & 0 \\ -m_{21} & \ddots & & 0 \\ \vdots & & \ddots & 0 \\ m_{n1} & \cdots & & m_{n,n-1} \end{bmatrix}$$

If there are no Ps, then $A = LU$, where L is lower-triangular, and U is upper-triangular.
We check if $PA = LU$, then does this mean $M_1P_1 = P_1M_1$? Recall that m_i below is a vector.

$$\begin{aligned}
M_1 &= I - m_1e_1^T, \qquad e_1^Tm_1 = 0 \\
P_1 &= I - (e_1 - e_j)(e_1 - e_j)^T \\
M_1P_1 &= \left(I - m_1e_1^T\right)\left((e_1 - e_j)(e_1 - e_j)^T\right) \\
&= I - (e_1 - e_j)(e_1 - e_j)^T - m_1\underbrace{\left[e_1^T(e_1 - e_j)\right]}_{1}(e_1 - e_j)^T - m_1e_1^T \\
&= I - (e_1 - e_j)(e_1 - e_j)^T + e_1e_j^T \\
P_1M_1 &= \left(I - (e_1 - e_j)(e_1 - e_j)^T\right)\left(I - m_1e_1^T\right) \\
&= I - m_1e_1^T - (e_1 - e_j)(e_1 - e_j)^T + (e_1 - e_j)(e_1 - e_j)^Tm_1e_1^T \\
&= I - (e_1 - e_j)(e_1 - e_j)^T - m_1e_1^T + (e_1 - e_j)e_j^Tm_1e_1^T
\end{aligned}$$

And we notice

$$m_1 + (e_1 - e_j)e_j^Tm_1$$

returns a vector like:

$$\begin{bmatrix} m_{j1} \\ m_{21} \\ \vdots \\ 0 \\ \vdots \\ m_{n1} \end{bmatrix},$$

where we replace the first entry by a possibly nonzero entry and insert a zero into that slot.
Hence we conclude that

$$P_1M_1 = M_1P_1$$

with m rearranged. Precisely, we write:

$$\begin{aligned}
U &= M_{n-1}P_{n-1}\cdots P_1A \\
&= \tilde{M}_{n-1}\tilde{M}_{n-2}\cdots\tilde{M}_1P_{n-1}\cdots P_1A \\
PA &= LU
\end{aligned}$$

2 Algorithm Without Pivoting

First we look at an algorithm without pivoting to see what we want to change.

```
for k = 1 : n-1 % k is column and row index
  for i = k+1 : n % row i
    % compute a_ik and check it is zero (or how close, for
    debugging)
    a_ik = a_ik / a_kk
    for j = k+1 : n
    % perform rank-1 update to bottom-right sub-matrix
      a_ij = a_ij - a_ik * a_kj;
    endfor
  endfor
endfor
```

3 Algorithm WITH Pivoting

First let's be smart and not actually perform row swaps but rather store an array with which row swaps to perform.

```
p = [1:n]
for k = 1:n+1
  [amax, imax ] = max( abs( a(k+1:n , k ) ) )
  imax = imax + k % fix the index of destination of swap
  iswap = p(k)
  p(k) = imax
  p(imax) = iswap

  for i = k+1 : n
    a_{p(i), k} = a_{p(i), k} / a_{p(k), k}
    for j = k+1:n
      a_{p(i), j} = a_{p(i),j} - a_{p(i),k} * a_{p(k),j}
    endfor
  endfor
endfor
```

Fortran stores values by columns, but some programming languages stores values by rows. It's beneficial to know how your data is stored in your programming language to control your memory storage and access.
Notice that Complete Pivoting (row and column swaps) is very expensive. We also have Rook pivoting (row and column swaps but not all), or Randomized Rook pivoting (for which Professor Gu is an authority).
After the break we'll look at diagonally dominant matrices (where pivoting won't be necessary), and later we'll look at positive definite matrices.

Break time.

4 Special Matrices: (1) Diagonally Dominant, (2) Symmetric Positive Definite

We consider two classes of matrices for which Gaussian elimination can be performed effectively without row interchanges (pivoting is not only not necessary but **possibly harmful**). The first is **diagonally dominant** matrices, and the second is **positive-definite** matrices. For symmetric positive-definite matrices, we take Cholesky decomposition.

4.1 Diagonally Dominant

Definition: Diagonally Dominant -

We say an $n \times n$ matrix A is **strictly** diagonally dominant when

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad \forall i = 1 : n$$

.

Take for example,

$$\begin{bmatrix} 10 & 1 & -2 \\ -3 & 42 & -4 \\ 50 & -6 & 630 \end{bmatrix}$$

Consider the solution to $Ax = b$ where:

$$x_i = b_i - \sum_{j\neq i} \overbrace{\left(\frac{a_{ij}}{a_{ii}}\right)}^{<1} x_j$$

involves an effective 'average' of x_j, and hence **fixed point iteration will converge**.

Theorem 4.1. If A is Diagonally Dominant, then Gaussian elimination **without** pivoting will succeed.

Proof. According to Strain, the fundamental rule in linear algebra is to induct on the dimension. In the 1×1 case, we have $ax = b$, with $a > 0$, and hence $L = 1$ and $U = a$, as desired.

For fun, consider the 2×2 case. For $a > |b|$ and $d > |c|$, check:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \to \begin{bmatrix} a & b \\ 0 & b - \frac{1}{a}cb \end{bmatrix}$$

and we check that $a \neq 0$. We already have $a > |b|$, so we consider:

$$d - \frac{b}{a}c > |c| - |c| = 0$$

Now we check the 3×3 case before declaring victory.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \to \begin{bmatrix} a & b & c \\ 0 & e - \frac{db}{a} & f - \frac{dc}{a} \\ 0 & h - \frac{gb}{a} & i - \frac{gc}{a} \end{bmatrix}$$

We want to check that the bottom-right 2×2 sub-matrix is diagonally dominant. To do this, we have $e > |f| + |d|$ and $a > |b| + |c|$. We check:

$$e - \frac{db}{a} > f - \frac{dc}{a}$$

by seeing:

$$\begin{aligned} e &> f - \frac{d}{a}c + \frac{d}{a}b \\ &= f + d\left(\frac{b}{a} + \frac{c}{a}\right) \leq f + d \end{aligned}$$

□

4.2 Symmetric Positive Definite Matrices

Last week, we discussed this type of matrix. (1) We have $A = A^T$ and $x^T Ax > 0$ unless $x = 0$. This requires us to test all possible x (which we sometimes can algebraically). This type is so important that we simply write:

$$A > 0,$$

which is **not** the same as saying all entries are positive. For example, take:

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0.$$

On the other hand, take:

$$\begin{bmatrix} x_1 x_2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} (2x_1 - x_2) \\ (-x_1 + 3x_2) \end{bmatrix} = 2x_1^2 - 2x_1x_2 + 3x_2^2 \geq x_1^2 + 2x_2^2 > 0$$

unless $x = 0$ because $2ab \leq a^2 + b^2$ from the trivial inequality on $(a + b)^2$.
(2) Another test for symmetric positive definite matrices ($A > 0$) is that if $\det(A_{kk}) > 0$ for all leading principle submatrices.
(3) Recall that the other test for $A > 0$ is that all the eigenvalues are strictly greater than 0. That is,

$$\lambda_j(A) > 0, \quad \forall j$$

Review Recall some classes of matrices:

Symmetric: $A = A^T$ (a symmetric matrix always has real eigenvalues.)
Normal (unitarily diagonalizable): $AA^T = A^T A$
Orthogonal: $A^T A = I = AA^T$
Projection: $A^2 = A$.

Now we want to show that a symmetric positive definite matrix (SPD) implies that no pivoting is necessary. Take the 2×2 case.

Lecture ends here.

Next time we'll look at Cholesky Factorizations.

Math 128A, Summer 2019
Lecture 27, Wednesday 8/7/2019

1 Review

We open with going over the first two questions in PSET 6. We have a predictor-corrector scheme. For $k = 1$, we have

$$\begin{aligned}
v_{n+1} &= u_n + hf_n, \qquad p(1) = 1 \\
u_{n+1} &= u_n + hf(t_{n+1}, v_{n+1}), \qquad q(1) = 1 \\
[f_{n+1} &= f(t_{n+1}, u_{n+1})], \qquad p = h\begin{bmatrix}\frac{3}{2} & \frac{-1}{2}\end{bmatrix} \\
v_{n+1} &= u_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right) \\
u_{n+1} &= u_n + h\left(\frac{1}{2}f(v_{n+1} + \frac{1}{2}f_n\right), \qquad q = h\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}
\end{aligned}$$

In `[p,q] = pcoeff(k,t,n)`, we technically can extract n from k, t; however, we don't want to program it this way, because we may have a very long vector, and n helps us know where we are in the vector.

2 Symmetric Positive Definite Matrices

This is a huge topic, and there are probably many books on this subject alone. Recall that the motivation behind this class of matrix is that we **don't need pivoting** for Gaussian elimination.
Suppose we have $x^T Ax > 0$, so that A is symmetric-positive-definite. Inspecting deeper,

$$A = \begin{bmatrix} a_{11} & b^T \\ b & A_{22} \end{bmatrix}$$

where A_{22} is a leading minor submatrix and must be also symmetric-positive definite.
From one step of Gaussian elimination,

$$M_1 A = \begin{bmatrix} a_1 1 & b^T \\ 0 & A_{22} - \frac{1}{a11} bb^T \end{bmatrix}$$

We want to show that one step of Gaussian elimination leaves the resulting matrix symmetric and positive definite. The problem here is that the matrix above is not symmetric! So we consider something that is structurally (destined to be) symmetric, $M_1 A M_1^T$.
Take M_1 as (and check):

$$M_1 = \begin{bmatrix} 1 & 0^T \\ -\frac{b}{a_{11}} & I \end{bmatrix}$$

and hence

$$M_1 A = \begin{bmatrix} 1 & 0^T \\ -\frac{b}{a_{11}} & I \end{bmatrix} \begin{bmatrix} a_{11} & b^T \\ b & A_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & b^T \\ 0 & A_{22} - \frac{1}{a_{11}} bb^T \end{bmatrix},$$

which is the result we had before, so our M_1 is correct. Now we evaluate:

$$M_1 A M_1^T = \begin{bmatrix} a_{11} & b^T \\ 0 & A_{22} - \frac{1}{a_{11}} bb^T \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{a_{11}} b^T \\ 0 & I \end{bmatrix} = \begin{bmatrix} a_{11} & 0^T \\ 0 & A_{22} - \frac{1}{a_{11}} bb^T \end{bmatrix},$$

and our question is if this is symmetric positive definite (we first check if it is symmetric).
The reason $bb^T \geq 0$ is that

$$x^T(bb^T)x = (x^Tb)(b^Tx) = (b^Tx)^2 \geq 0.$$

Recall that $A_{22} > 0$ (submatrix of symmetric positive definite). Let $y \in \mathbb{R}^{n-1}$, so that

$$y^T\left(A_{22} - \frac{1}{a_{11}}bb^T\right)y,$$

and augment matrices to have:

$$\underbrace{\begin{bmatrix}0 & y^T\end{bmatrix}}_{x^T}\underbrace{\begin{bmatrix}a_{11} & 0\\ 0 & A_{22}-\frac{1}{a_{11}}bb^T\end{bmatrix}}_{M_1AM_1^T}\underbrace{\begin{bmatrix}0\\ y\end{bmatrix}}_{x\in\mathbb{R}^n} = x^TM_1AM_1^Tx = \left(M_1^Tx\right)^TA\left(M_1^Tx\right) > 0,$$

particularly if $y \neq 0$.
Now $A > 0$ implies

$$M_1AM_1^T = \begin{bmatrix}a_{11} & 0^T\\ 0 & A_{22}-\frac{1}{a_{11}}bb^T\end{bmatrix} = \begin{bmatrix}a_{11} & 0^T\\ 0 & M_2\begin{bmatrix}a_22' & 0^T\\ 0 & A'_{33}-\frac{1}{a'_{11}}b'b'^T\end{bmatrix}M_2^T\end{bmatrix}$$

$$= \begin{bmatrix}1 & \\ & M_2\end{bmatrix}\begin{bmatrix}\begin{bmatrix}a_{11} & 0\\ 0 & a_{22}\end{bmatrix} & 0^T\\ 0 & \underbrace{A''_{33}}_{>0}\end{bmatrix}\begin{bmatrix}1 & \\ & M_2^T\end{bmatrix}$$

So we deduce:

$$M_{n-1}\cdots M_1AM_1^T\cdots M_{n-1}^T = \begin{bmatrix}d_{11} & & 0\\ & \ddots & \\ 0 & & d_{nn}\end{bmatrix}$$

which we call this
Symmetric unit-lower-triangular reduction to diagonal form, also known as LDL^T factorization or **Cholesky factorization**.
That is, we have

$$A = LDL^T,$$

where D has entries $d_{ii} > 0$. And if $A = A^T$, it's plausible to say we have a theorem:

Theorem 2.1.

$$\begin{aligned}A = A^T > 0 &\iff x^TAx > 0 \forall_{x\neq 0}\\ &\iff A = L^TDL, \quad L \text{ is unit lower-triangular, } D > 0 \text{ diagonal}\\ &\iff \exists_{B \text{ invertible}} A = B^TB\\ &\iff \exists_{R \text{ upper-triangular}} A = R^TR \quad (R^TR)\end{aligned}$$

After the break we'll talk about 3 different ways to compute the Cholesky factorization.

Break time.

3 Cholesky Factorization

This is symmetric Gaussian Elimination without Pivoting. Because we know we are not pivoting, we are really just solving a bunch of equations for a symmetric matrix:

$$A = R^T R$$

$$\begin{bmatrix} \ddots & & \\ & a_{ij} & \\ & & \ddots \end{bmatrix} = \begin{bmatrix} r_{11} & & 0 \\ & \ddots & \\ r_{1n} & & r_{nn} \end{bmatrix}$$

Getting these matrices satisfied is just $\frac{n(n_1)}{2}$ quadratic equations. Due to the ('symmetric') form of A and correspondingly the lower-triangular times upper-triangular on the RHS, we have a chance at finding an order in which we can isolate and solve for one at a time. That is,

$$\begin{aligned} a_{11} &= r_{11}^2 \\ a_{12} &= r_{11} r_{12} \\ &\vdots \\ a_{1n} &= r_{11} r_{1n} \end{aligned}$$

which gives us a full row of R and a column of R^T, so we have a lot of information already! Next we have:

$$\begin{aligned} a_{22} &= r_{12}^2 + r_{22}^2 \\ a_{23} &= r_{12} r_{13} + r_{22} \underbrace{r_{23}}_{\text{only unknown}} \\ &\vdots \end{aligned}$$

Example: Our favorite symmetric positive definite matrix.

Actually, we cheat by taking:

$$\begin{bmatrix} 1 & 0 \\ -2 & 3 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ -2 & 13 \end{bmatrix} > 0$$

and we constructed a symmetric positive definite matrix. Now how do we find its Cholesky factorization?

$$\begin{bmatrix} 1 & -2 \\ -2 & 13 \end{bmatrix} = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} = \begin{bmatrix} a^2 & ab \\ ab & b^2 + c^2 \end{bmatrix},$$

which gives us the equations:

$$\begin{aligned} a^2 = 1 &\implies a = 1 \\ ab = -2 &\implies b = -2 \\ b^2 + c^2 = 13 &\implies c = 3, \end{aligned}$$

which gives us our original matrices.

3.1 Can Finding the Cholesky Factorization Fail?

It turns out, this only fails if A is not Symmetric-Positive-Definite.
If this fails, given R, insert E such that $(R+E)$ satisfies the Cholesky factorization:

$$(R+E)^T(R+E) = A$$

This gives:

$$\begin{aligned} R^TR + R^TE + E^TR + E^TE &= A \\ R^TE + E^TR &= A - R^TR - \underbrace{\cancel{E^TE}}_{O(E^2)} \end{aligned}$$

This is one step of Newton's method, where we are linearizing and throwing away the quadratic term. We know that Newton's converges quadratically when it works.

$$\begin{aligned} R^TER^{-1} + E^T &= AR^{-1} - R^T \\ \underbrace{E}_{up.tri.}\ \underbrace{R^{-1}}_{up.tri.} + \underbrace{R^{-T}}_{low.tri}\ \underbrace{E^T}_{low.tri} &= \underbrace{R^{-T}AR^{-1} - I}_{\text{symmetric}} \end{aligned}$$

Note that E and R^{-1} are each upper-triangular and R^{-T}, E^T are each lower-triangular.

Definition: Causality -

Take a lower-triangular matrix L and a time vector $T = \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix}$, where

$$\begin{bmatrix} \cdot & & \\ \cdot & \cdot & \\ & & \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \\ \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \end{bmatrix}$$

Hence we say:

$$ER^{-1} = \text{uph}(R^{-T}AR^{-1} - I), \tag{1}$$

where

$$\text{uph}(A)_{ij} := \begin{cases} A_{ij}, & j > i \\ \frac{1}{2}A_{ij}, & j = i \\ 0, & j < i \end{cases},$$

where $j = i$ gives the upper triangular with half on the diagonal. So E solves the following equation:

$$(R+E)^T(R+E) = A + E^TE \underbrace{(R+E)^T(R+E) - A}_{\text{residual}} \qquad = E^TE$$

Now we want to express a relationship between the residual and the error. We write:

$$E := \text{uph}(R^T A R^{-1} - I)R$$

from (1) above.

and hence

$$\begin{aligned} E^T E &= \left(\text{uph}(R^T A R^{-1} - I)R\right)^T \left(\text{uph}(R^{-T} A R^{-1} - I)R\right) \\ &= R^T \text{uph}(R^{-T} \underbrace{[A - R^T R]}_{\text{old residual}} R^{-1}) \cdot \text{uph}(R^{-T} \underbrace{[A - R^T R]}_{\text{old residual}} R^{-1})R \end{aligned}$$

$$\frac{E^T E}{A} = \underbrace{O\left((A - R^T R)^2\right)}_{A^2} \underbrace{R^{-2}}_{A^{-1}},$$

and so we say relative residual is squaring.

Lecture ends here.

Math 128A, Summer 2019
Lecture 28, 8/8/2019

1 Review: Homework 6

1.1 Problem 5

Consider the Fredholm equation:

$$u(x) + \int_0^1 \underbrace{K(x,y)}\, u(y)\ dy = g(x)$$

which is a rank 1 perturbation of the identity.
versus the Volterra equation:

$$y(t) = y(0) + \int_0^t \underbrace{f(s, y(s))}_{\text{causal}}\ ds$$

and $x_i + \sum_{j=1}^n A_{ij}x_j = b_i$ which is $(I + A)x = b$ so $x = b - Ax$.
Fredholm are generally harder than Voltera, but in our problem we have a special kernel (K for kernel),

$$K(x, y) = \cos(x)\sin(y)$$

with

$$\int_0^1 \cos(x)\sin(y)u(y)\ dy = \cos(x)\int_0^1 \sin(y)u(y)\ dy$$

We need to solve forward and backward simultaneously. Recall that any rank-1 matrix, say A, is a row vector left-multiplied by a column vector:

$$A = ab^T = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} b_1 & \cdots & b_n \end{bmatrix} = \begin{bmatrix} a_1b_1 & \cdots & a_1b_n \\ \vdots & & \vdots \\ a_nb_1 & \cdots & a_nb_n \end{bmatrix}$$

Then let $(I + A)x = v$, so that:

$$\begin{aligned} x + a(b^Tx) &= v \\ x &= v - \underbrace{(b^Tx)}_{\text{scalar}}\, a \end{aligned}$$

We have an idea: let's take the dot product across the equation with b.

$$\begin{aligned} b^Tx &= b^Tv - (b^Tx)(b^Ta) \\ (1 + b^Ta)b^Tx &= b^Tv \\ b^Tx &= \frac{b^Tv}{1 + b^Ta}, \end{aligned}$$

if $b^Ta \neq -1$. This is how we solve this rank-1 perturbation problem. In this particular problem, we have a rank-1 kernel, and thus our problem has a rank-1 perturbation.
In our problem, once we discretize the equation via gaussian quadrature, we take on the linear system to solve via Gaussian elimination with partial pivoting (GEPP).

2 Review

Yesterday, we took A symmetric positive definite, with $x^TAx > 0$ for $x \neq 0$ and $A = A^T$. We computed the Cholesky factorization,

$$A = R^TR = LL^T,$$

where R is (right) upper-triangular with $r_{ii} > 0$, we check:

$$\begin{aligned} x^TAx &= (Rx)^T(Rx) \\ &= \|Rx\|^2 > 0, \end{aligned}$$

if $x \neq 0$, because R is invertible.
One of the checks if A is symmetric positive definite is for us to compute the cholesky factorization, and if multiplying them together doesn't return A, then A is not symmetric positive definite (we have nothing to lose).
Recall that we have three ways to compute the Cholesky factorization:
(1) $LDL^T = \underbrace{(\sqrt{D}L^T)^T}_{R^T}\underbrace{(\sqrt{D}L^T)}_{R}$
(2) Direct calculation, solving for one element at a time (and using symmetry). We specify an algorithm to do this. First we write the underlying math:

$$i \geq i: \quad a_{ij} = \sum_{k=1}^{i} r_{ki}\, \underbrace{r_{kj}}$$

where $r_{kj} = r_{1j}r_{1j} + r_{2i}r_{2j} + \cdots + r_{ii}\overbrace{r_{ij}}$ where r_{ij} is what we solve for first, then the rest one at a time.
Additionally,

$$a_{ii} = \sum_{k=1}^{1} r_{ki}^2$$

so we find a_{11} first: $r_{11} = \sqrt{a_{11}}$.
and note $r_{kj} = 0$ when $k > j$ and $r_{ki} = 0$ when $k > i$. Our pseudocode is as follows: Find column j of $\mathbb{R}$.

```
for j = 1:n
  for i = 1:j-1
    r_{ij} = ( a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} ) / r_{ii
    }
  end
  r_{jj} = \sqrt{ a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2 }
end
```

(3) Use Newton's Method for the function we are trying to solve:

$$f(R) = R^TR - A = 0.$$

and of course, R is an upper-triangular matrix with nonnegative diagonal entries. We can say we want to find a correction E such that:

$$(R+E)^T(R+E) - A = E^TE,$$

where we usually set this equal to 0, but instead we cancel out the quadratic term. We check the case that if this is exact, then we would have to deal with a quadratic term, so we insert (recall from yesterday):

$$E := \text{uph}(R^TAR^{-1} - I)R$$

On the other hand, if F is the exact correction to the Cholesky factorization (and E is only what we can get),

$$(R+F)^T(R+F) - A = 0$$

Comparing these,

$$\begin{aligned} R^TR - A + E^TR + R^TE + E^TE &= E^TE \\ R^TR - A + F^TR + R^TF + F^TF &= 0 \end{aligned}$$

where subtracting gives

$$(E-F)^TR + R^T \underbrace{(E-F)}_{(R+E)-(R+F)} = F^TF$$

and rewriting,

$$\underbrace{R^T}_{low.tri}\underbrace{(E-F)^T}_{low.tri} + \underbrace{(E-F)}_{up.tri}\underbrace{R^{-1}}_{up.tri} = R^{-T}F^TFR^{-1},$$

where a lower-triangular times a lower-triangular matrix gives a lower triangular matrix and accordingly for upper-triangular matrices. Hence this equation takes the form of a lower-triangular matrix plus an upper-triangular matrix equals a full matrix.
Hence the error in the correction, the error in $(R+E)$ since $(R+F)$ is exactly Cholesky factor.
We have, for the error relative to the solution we started with,

$$\begin{aligned} (E-F)R^{-1} &= \text{uph}\left((FR^{-1})^T(FR^{-1})\right), \\ E-F &= \text{uph}\left(R^{-T}\underbrace{F^TF}_{(\text{err in } R)^T\cdot(\text{err in } R)}R^{-1}\right)R \end{aligned}$$

which is in terms of the relative error of the approximate old solution, so is **quadratic convergence** a la **Newton's**.

3 Floating Point Errors in Cholesky Factorization

When we compute the Cholesky factorizations,

$$fl(r_{11}) = fl(\sqrt{a_{11}}) = \sqrt{a_{11}}(1+\delta_{11}),$$

where $|\delta_{11}| \le \varepsilon$ is the exact result, correctly rounded. This gives:

$$\begin{aligned} r_{11}^2 &= a_{11}(1+\delta_{11})^2 \\ &= a_{11}(1+2\delta_{11}) + \cancel{O(\varepsilon^2)} \\ &= \hat{a}_{11}, \end{aligned}$$

where $|\hat{a}_{11} - a_{11}| \le 2\varepsilon|a_{11}|$. Our hope is that

$$\hat{R} = fl(R)$$

is the **exact** Cholesky factor of

$$\left|\hat{A} - A\right| \le O(n)\varepsilon|A| = [|a_{ij}|].$$

Because we don't yet have the notion of a norm, we check each entry value and check that each can be bounded.
Remember from our algorithm earlier for directly computing the Cholesky factorization,

$$r_{ij} = \left(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj} \right) / r_{ii}$$

where we can only work with what we know ('the guys with hats'). Recalling $fl(a-b) = a(1+\delta) - b(1+\delta)$, what actually happens is:

$$\hat{r}_{ij} = \left(a_{ij}(1+\delta_{ij}) - \sum_{k=1}^{i-1} \hat{r}_{ki}\hat{r}_{kj}(1+i\delta_{ij}) \right) / (r_{ii}(1+\delta_{ii}))$$

We tend to not like forward error analysis (as the above). We prefer backwards error analysis as the computed results almost satisfying the exact equation it is trying to solve.
Multiplying across by the denominator gives:

$$\hat{r}_{ii}\hat{r}_{ij}(1+\delta''_{ii}) = a_{ij}(1+\delta_{ij}) - \sum_{k=1}^{i-1} \hat{r}_{ki}\hat{r}_{kj} - \sum_{k=1}^{i-1} \hat{r}_{ki}\hat{r}_{kj} i d_{ij}$$

What we have is doing exactly what we want, if we interpret everything a little differently.

$$\hat{r}_{ii}\hat{r}_{ij} = \underbrace{\left[a_{ij}(1+\delta_{ij}) - \sum_{k=1}^{i} \hat{r}_{ki}\hat{r}_{kj} i\delta_{ij} \right]}_{\text{scapegoat}} - \sum_{k=1}^{i-1} \hat{r}_{ki}\hat{r}_{kj}$$

and this is the exact Cholesky factor of our matrix, where we labeled the bracketed quantity as the 'scapegoat' as the source of our error.
So we conclude:

$\hat{R}$ is the exact Cholesky factor of $\hat{A} = \left[a_{ij}(1+\delta_{ij}) - \sum_{k=1}^{i} \hat{r}_{ki}\hat{r}_{kj} i\delta_{ij} \right]$, and we write the error as a good quantity plus an uncertain quantity:

$$|\hat{A} - A| \leq \varepsilon |A| + (n\varepsilon)|\hat{R}^T||\hat{R}|,$$

where $|M|$ denotes taking the absolute value of every element.

This means that all we need to do is take the absolute value of all entries, place them into a matrix R, multiply it by its transpose, then we get close to the exact Cholesky factors.
We may also want to check how close is $\hat{R}$ to **the exact** cholesky factor? Then, what can we say about how close $\hat{R}$ is to R. This gives a measure of **how sensitive** the Cholesky factors are to **perturbations** in A.

4 Sensitivity to Perturbations

Generally we look at the sensitivity of the inverse A^{-1} by perturbations in A, say $A \mapsto A + E$. We write per element, via geometric series:

$$\frac{1}{a+e} = \frac{1}{a\left(1+\frac{e}{a}\right)} = \frac{1}{a}\left(1 - \frac{e}{a} + O(\frac{e}{a})^2\right)$$

Now how do we do this for matrices?

$$(A+E)^{-1} = \left(A(I+A^{-1}E)\right)^{-1}$$

We need a geometric series for matrices. Recall the derivation for the 1-dimensional case:

$$\begin{aligned}
\frac{1-x^{n+1}}{1-x} &= 1+x+x^2+\cdots+x^n \\
1-x^{n+1} &= (1-x)(1+x+\cdots+x^n) \\
&= (1+x+\cdots+x^n) - \left(x+x^2+\cdots+x^{n+1}\right) \\
&= 1-x^{n+1}.
\end{aligned}$$

and in the matrix case,

$$\begin{aligned}
\left(I-A^{n+1}\right)(I-A)^{-1} &= I+A+\cdots+A^n \\
\left(I-A^{n+1}\right) &= (I+A+\cdots+A^n)(I-A) \\
&= I+A+\cdots+A^n - \left(A+\cdots+A^{n+1}\right)
\end{aligned}$$

The first condition for an infinite geometric series of matrices is that $(I-A)^{-1}$ must **exist**, and the second is that $A^n \xrightarrow{n\to\infty} 0$. It turns out this is equivalent ($\iff$) all eigenvalues of A are < 1 in absolute values: $|\lambda_j(A)| < 1, \forall j$.

Lecture ends here.

Next time, we'll think of a way to prove this **without** using Jordan Normal Form. We'll talk about **perturbation theory** and perhaps **norms**.

Math 128A, Summer 2019

Lecture 29, 8/12/2019

Plans for today:
- Matrix Norms
- Iterative Improvement

1 Floating Point Error: $Ax = b$

Suppose we want to solve the linear system, $Ax = b$. We don't know if this is solvable, but we have technology to try. The first step is to try to factorize:

(1) $PA = LU$
(2) Use the above permutation to solve $Ax = b$. Let $Pb = PAx = c$. Permute the entries of b
(3) Now let $Ly = c$
(4) Let $Ux = y$.

There are quite a few steps here; however, we sometimes cannot just 'divide' by A. Then $c = LUx = Pax$. The advantage here is that these steps are simple and routine, and a computer can do it.

For example, consider $Ly = c$ for (unit)-lower-triangular L:

$$\begin{bmatrix} l_{11} & & & 0 \\ l_{21} & l_{22} & & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \\ \vdots \\ c_n \end{bmatrix}$$

Solving this lower-triangular system gives:

$$\begin{aligned} y_1 &= c_1/l_{11} \\ y_2 &= (c_2 - l_{21}y_1)/l_{22} \\ &\vdots \\ y_n &= (c_n - l_{n1}y_1 - l_{n2}y_2 - \cdots - l_{n,n-1}y_{n-1})/l_{nn} \end{aligned}$$

1.1 Backwards Error Analysis: Floating Point Arithmetic

Of course, we get the exact result (from operations), correctly rounded **at each step of operations**. However, errors do propagate down the system. We write in terms of each slightly perturbed data values:

$$\begin{aligned} \hat{y}_1 = fl(y_1) &= (c_1/l_{11})(1+\delta_1), \qquad |\delta| \le \frac{\varepsilon}{2} \\ fl(y_2) = \hat{y}_2 &= \left((c_2 - l_{21}\hat{y}_1(1+\delta_1))(1+\delta_2)/l_{22}\right)(1+\delta_3) \\ &= \left[c_2(1+\delta_3)(1+\delta_2) - \underbrace{l_{21}(1+\delta_1)(1+\delta_2)(1+\delta_3)}_{\text{blame this for error}}\hat{y}_1\right]/l_{22} \\ &= \hat{c}_2 - \hat{l}_{21}\hat{y}_1/l_{22}, \end{aligned}$$

where we strategically let the numerators absorb the errors, and we leave l_{22} exact.
So we can write:

$$\hat{y}_n = \left(\hat{c}_n - (\hat{l}_n\hat{y}_1 + \cdots + \hat{l}_{n,n-1}\hat{y}_{n-1})\right)/l_{nn}$$

How close are these terms? We can say that $\hat{c}_n$ participates in each of the subtractions, so

$$|\hat{c}_n - c_n| \leq \varepsilon|c_n|,$$

and we conclude:

$$\left|\hat{l}_{ij} - l_{ij}\right| \leq n\varepsilon|l_{ij}|$$

In other words, in terms of matrices and vectors, for $Ly = c$ and $\hat{L}\hat{y} = \hat{c}$, we say that for each value within vector c and matrix L, we have, :

$$|\hat{c} - c| \leq \varepsilon|c|, \qquad |\hat{L} - L| \leq n\varepsilon|L|,$$

so $L_{ij} = 0 \implies \hat{L}_{ij} = 0$. This gives us our desired **backwards error analysis**.

We've shown that the computed solution is the exact solution to a slightly perturbed matrix $\hat{L}$. The backwards error is usually enough, to say that we solved a problem that is close to the original problem. However, we may need to explain (to our boss) why things went wrong, and we'll need forward error analysis. Strain says that backwards error analysis implies forward error analysis.

1.2 Forward Error Analysis

$$\begin{aligned} Ly = \hat{L}\hat{y} &= c - \hat{c} \\ Ly - L\hat{y} + L\hat{y} - \hat{L}\hat{y} &= c - \hat{c} \\ L(y - \hat{y}) + (L - \hat{L})y &= c - \hat{c} \\ L(y - \hat{y}) &= c - \hat{c} - (L - \hat{L})\hat{y} \end{aligned}$$

On the right hand side, we have the perturbation minus the perturbation in the matrix, applied to the computed solution. Now if we are given a reasonable problem to solve, then L will be invertible, so :

$$y - \hat{y} = L^{-1}$$

This is an exact formula and contains some inverses, so we want to have some generalized bound (inequality):

$$|y - \hat{y}| \leq |L^{-1}(c - \hat{c})| + |L^{-1}(L - \hat{L})\hat{y}|$$

This is more quantitative than saying we solved almost the right problem. To get even more quantitative, we say:

$$\begin{aligned} |(Ax)_i| &\leq \sum_j |A_{ij}x_j| \\ &\leq \sum_j |A_{ij}||x_j| \\ &= (|A||x|)_i \end{aligned}$$

Let $|L^{-1}|$ to be the resulting matrix by replacing all elements by each of their absolute values.
So we can say, with respect to the first inequality:

$$\begin{aligned}|y-\hat{y}| &\leq |L^{-1}(c-\hat{c})| + |L^{-1}(L-\hat{L})\hat{y}| \\ &\leq \varepsilon|L^{-1}||c| + n\varepsilon|L^{-1}||L||\hat{y}| \\ &\leq \varepsilon \underbrace{|L^{-1}||L|}|y| + n\varepsilon \underbrace{|L^{-1}||L|}|\hat{y}| \\ &\leq \boxed{(n+1)\varepsilon|L^{-1}||L||y| + n\varepsilon|L^{-1}||L||y-\hat{y}|}\end{aligned}$$

where $|c| = |Ly| \leq |L||y|$, so we tack on the penultimate simplification. It's important to notice that $|L^{-1}||L| > I$ (always larger than the identity matrix). To get the last inequality, we try writing: $\hat{y} := y - (y - \hat{y})$.
Another way to write this final bound is:

$$\left(I - n\varepsilon|L^{-1}||L|\right)|y-\hat{y}| \leq (n+1)\varepsilon|L^{-1}||L||y|,$$

and we want to multiply by the inverse. However, will multiplying by a matrix preserve monotonicity of the inequality chain? We may flip some (or all) of the inequalities. One way to get around this problem is adding all these inequalities together and multiply across by a positive value. We conclude that
Recall from lecture 28, we worked out the geometric series of matrices. We found:

$$(I-A)^{-1} = I + A + A^2 + \cdots,$$

if $|\lambda_j(A)| < 1$ for all $j \in \mathtt{1:n}$. So, if

$$A \geq 0, A^2 \geq 0, A^3 \geq 0, \cdots,$$

then

$$\left(I - n\varepsilon|L^{-1}||L|\right)^{-1}_{ij} \geq 0,$$

as long as

$$\boxed{n\varepsilon\left|\lambda_j(|L^{-1}||L|)\right| < 1}$$

We conclude:

$$\begin{aligned}|y-\hat{y}| &\leq \left(I - \underbrace{n\varepsilon|L^{-1}||L|}_{=:A}\right)^{-1}(n+1)\varepsilon|L^{-1}||L||y| \\ &= \left(I + n\varepsilon|L^{-1}||L| + O(n^2\varepsilon^2)\right)(n+1)\varepsilon|L^{-1}||L||y|,\end{aligned}$$

so

$$|y-\hat{y}| \leq (n+1)\varepsilon|L^{-1}||L||y| + O(n^2\varepsilon^2),$$

if $n\varepsilon\lambda_j(|L^{-1}||L|) < 1$.
We can pretty much 'guesstimate' how large this error is by looking at $|L^{-1}|$. So if we get an incorrect, then we say that our model is slightly incorrect (we were handed a bad problem).
Notice that we didn't use the fact that L is lower-triangular, so this is true for all matrices A. Additionally, this applies to if y is not a vector but rather

a matrix. This would tell us about the inverse of a matrix, provided that the Cholesky factors are under control. Recall that in our present case, L has 1s on its diagonal and m_{ij} (multipliers) lower-left entries come from:

$$|m_{ij}| = \left|\frac{a_{ij}}{a_{jj}}\right| \leq 1$$

After the break, we'll talk about matrix norms. Likely tomorrow we'll look at iterative improvement.

Break time.

2 Matrix Norms

Now we want to simplify our previous inequalities which depended on the entire matrices $|L^{-1}|$ and $|L|$. First off, recall the three norms of a vector.

2.1 Review: Vector Norms $||x||_p$

(1) $p = 1$:

$$||x||_p = \sum_{j=1}^{n} |x_j|$$

(2) Euclidean norm ($p = 2$):

$$||x||_p = \sqrt{\sum_{j=1}^{n} x_j^2}$$

(3) $p = \infty$:

$$||x||_p = \max_j |x_j|$$

(Actually, there is a family of norms for different values of p, but we only care about these three for our purposes.)

In the euclidean norm, the 'unit ball' is the standard unit circle. In the max case ($p = \infty$), the 'unit ball' is the unit square. For $p = 1$, the 'unit ball' is the unit diamond.

Hence we say:

$$\boxed{||x||_\infty \leq ||x||_2 \leq \underbrace{||x||_1} \leq \sqrt{n}||x||_2}$$

It turns out we can say more than this, to get the underbraced inequality:

$$\begin{aligned}
||x||_1 &= \sum_{j=1}^{n} |x_j| \cdot 1 \\
&= |x|^T e, \qquad e = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \\
&= ||x||_2 ||e||_2, \qquad \text{Cauchy-Schwarz}, \\
&\leq \sqrt{n}||x||_2,
\end{aligned}$$

where we get the last inequality from $||e||_2 = \sqrt{\underbrace{1+1+\cdots+1}_{n}} = \sqrt{n}$.
We say that all norms are (topologically) **equivalent** in finite-dimensional spaces, in the sense that a sequence converging to 0 in one norm converges to 0 in another norm.

2.2 Matrix Norm

We say that Matrix norms take off from vector norms on a spring-board. Consider A as an operator on a vector $\vec{x}$ (we will simply write x). We may not know how big A is or x is, but we can look at

$$\frac{||Ax||}{||x||}$$

and take the worst case over all x. That is, we can take:

$$||A||_{2,1,\infty} := \max_{x \neq 0} \frac{||Ax||_{2,1,\infty}}{||x||_{2,1,\infty}}$$

Of course, this is not actually how we compute these, because this would require us to check every single x.
Facts:

$$||A_1|| = \max_j \sum_{i=1}^{n} |a_{ij}|,$$

where we take the columns, sum their values, and take the max of these. For example,

$$\left\| \begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \\ 7 & -8 & 9 \end{bmatrix} \right\|_1 = \max 12, 15, 18 = 18$$

The dual to this is summing the rows:

$$||A||_\infty := ||A^T||_1 = \max_i \sum_{j=1}^{n} |a_{ij}|,$$

where in our example, the ∞-norm of that matrix is 24.
On the other hand,

$$||A||_2 := \max \text{ eigenvalue of } A^T A = \lambda_{\max}(A^T A)$$

And a weird (but true!) fact is:

$$\boxed{||A||_2^2 \leq ||A||_1 ||A||_\infty},$$

which is worth mentioning because each of the norms on the RHS is easy to compute, whereas computation for $||A||_2^2$ can be difficult or intensive. It's good to know there's an easy-to-compute upper bound for this.
We've found a general class of norms, which we call **operator norms**. The last 1 of 4 norms we consider is **not** an operator norm:

2.3 Frobenius norm

$$||a||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}|^2}$$

We take the absolute value and square in the case that we are over the complex field $\mathbb{C}$, but in our purposes in 128A, we just work in $\mathbb{R}$. Recall the ways we derived to find the Cholesky factorization of a matrix.
(1) Gaussian elimination
(2) Equate all terms and solve a bunch of equations.
(3) Use Newton's method, take an initial guess and update using a matrix uph (upper-triangular, halved)

$$uph(A)_ij := \begin{cases} a_{ij}, & i > j \\ \frac{1}{2}a_{ij}, & i = j \\ 0, & i > j \end{cases}$$

We want the Frobenius norm for the third method above.

Definition: Absolute norm -

We say that a norm is an **absolute norm** if it depends only on the absolute values of the entries.

Surely, the 2-norm is **not** an absolute norm. This is important because we know that when we solve $Ly = c$ in floating point and we get $\hat{y}$ which solves a nearby problem $\hat{L}\hat{y} = \hat{c}$ where $|c - \hat{c}| \leq \varepsilon|c|$ and $|L - \hat{L}| \leq n\varepsilon|L|$, then we proved earlier that:

$$|y - \hat{y}| \leq (n+1)\varepsilon|L^{-1}||L||y| + O(n^2\varepsilon^2).$$

In terms of matrices, if we have an **absolute norm**, then we can simply convert single bars (absolute values) into double bars (norms). Because of the way norms are defined,

$$||Ax|| \leq ||A|||x||,$$

because we took $||A||$ as the maximum of the fraction $\frac{||Ax||}{||x||}$. This is interesting because if we take an eigenvalue with $Ax = \lambda x$, then we have:

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

which tells us:

$$|\lambda| \leq ||A||,$$

where in the 2-norm, we can have equality, but in general we have strict inequality. This means that, for example, if we're looking at an inverse and we want to take the geometric series to expand it, then if $||A|| \leq \frac{1}{2}$, the geometric series works really well. We say this in the same sense that every term of this sequence 'ought to give us an additional bit of accuracy, otherwise we won't want to use it'. Technically, if we have $||A|| < 1$, then this converges (but might be too slow at doing so).

Hence if we take two matrices A, B and multiply them together, we have:

$$\begin{aligned} ||ABx|| &\leq ||A||\ ||Bx|| \\ &\leq ||A||\ ||B||\ ||x||, \end{aligned}$$

and so (for operator norms which are sub-multiplicative):

$$||AB|| \leq ||A||\ ||B||$$

This is why we can do calculations like:

$$|y - \hat{y}| \leq (n+1)\varepsilon |L^{-1}||L||y|,$$

where when we take the norm of a vector, this is simply an absolute norm of the vector. So we write:

$$\frac{||y - \hat{y}||}{||y||} \leq (n+1)\varepsilon \underbrace{||\ |L^{-1}||L|\ ||}_{\text{a number to blame error}},$$

where we can take advantage of the fact that we can divide by the norm $||y||$. Let

$$\boxed{K_{CR}(L) := ||\ |L^{-1}||L|\ ||}$$

be the ***component-wise relative* condition number** of L. If this is is close to 1, within $O(1)$, then we should have no issues and can solve simply. If this is large, then we can look deeper into the problem (or seek to be paid overtime).
Recall from the MVT, we have the following 'condition number'

$$\frac{f(x) - f(y)}{f(x)} = \boxed{\frac{xf'(\xi)}{f(x)}} \cdot \frac{(x-y)}{x},$$

where $\frac{x-y}{x}$ is the relative change in the input, and on the left-hand-side, $\frac{f(x)-f(y)}{f(x)}$ is the relative change in the output. The boxed quantity is the condition number of evaluating f at x, which is an estimate of how 'scary' evaluting near a point is. Immediately, we can see that if we are computing zero and the derivative is nonzero, our condition number is high.
Important: If we derive the above carelessly, we can get get the following '**condition number**' which is a loose bound:

$$\kappa(L) := ||L^{-1}||\ ||L||$$

Notice $\kappa(L) > K_{CR}(L)$.

Lecture ends here.

Tomorrow, we'll look at iterative improvement.

Math 128A, Summer 2019
Lecture 30, Tuesday , 8/13/2019

1 Review: (Past) Final Exam Questions

Recall from 2019, we have the following problem:

$$\int_0^1 f(t)\ dt = af(1) + bf(0) + cf(-\theta),$$

for $\theta \in [-\frac{1}{2}, 2]$. The zeroth question is what is the highest degree polynomial f for which this integration rule will be exact? Turns out we can do so for degree up to 2.
Taking $f(t)$ to be $1, t, t^2$, we get the following system of equations (via the monomial basis):

$$\begin{aligned} 1 &= a + b + c \\ \frac{1}{2} &= a \cdot 1 + b \cdot 0 + c \cdot (-\theta) \\ \frac{1}{3} &= a \cdot 1^2 + b \cdot 0^2 + c(-\theta)^2, \end{aligned}$$

or equivalently we can look at the Newton basis (with malice of forethought to give a triangular system):

$$f(t) = f_0 + f_1(t) + f_2 t(t + \theta),$$

with

$$\begin{aligned} f_0 &= 1 \\ f_1 &= t \\ f_2 &= t(t + \theta) \end{aligned}$$

to give

$$\begin{aligned} \int_0^1 1dt = 1 &= a + b + c \\ \int_0^1 t\ dt = \frac{1}{2} &= a + b \cdot 0 + c(-\theta) \\ \int_0^1 t(t + \theta)\ dt = \frac{1}{3} + \frac{\theta}{2} &= a \cdot 1(1 + \theta) + b \cdot 0 \cdot (0 + \theta) + c(-\theta)(-\theta + \theta) \end{aligned}$$

or equivalently, we can use the Lagrange basis (diagonal linear system), although solving the integrals can be

1.1 3B

Suppose we have an **implicit variable-step Adams method**, where:

$$\begin{aligned} t_n - t_{n-1} &= \theta h \\ t_{n+1} - t_n &= h, \end{aligned}$$

we and an integration formula to take t_n to t_{n+1} into 0 to 1:

$$y(t_{n+1}) = y(t_n) + \underbrace{\int_{t_n}^{t_{n+1}} y'(s)\ ds} = f(s, y(s))\ ds,$$

and so transform this into

$$h\int_0^1 y'(t_n + th_1)\ dt = h[ay'(t_{n+1}) + by'(t_n) + cy'(t_{n-1})]$$

This means that our Adams method is

$$u_{n+1} = u_n + h[af_{n+1} + bf_n + cf_{n-1}]$$

Take the interpolating polynomial for y' :

$$p(t_{n+j}) = f_{n+j}, \qquad j = 1, 0, -1$$

Calculating the truncation error is simply:

$$h\tau_{n+1} := \int_{t_n}^{t_{n+1}} y'(s) - p(s)\ ds,$$

where we can use taylor series to expand; however in this case we know everything there is to know about polynomial interpolation, set $w := y'$, so:

$$\begin{aligned} w(s) - p(s) = y'(s) - p(s) &= \frac{w'''(\xi)}{3!}(s - s_{n+1})(s - s_n)(s - s_{n-1}) \\ &= \frac{y^{(4)}(\xi_s)}{3!}[(s - s_{n+1})(s - s_n)(s - s_{n-1})] \end{aligned}$$

(As an aside, the dimensions of local truncation error (in euler) indicate that it is a measure of error in y', not y. That is, $\tau_{n+1} = \frac{y_{n+1} - y_n}{h}$.)
We want to pull the derivative factor out, and this is legal precisely as long as $\omega(t)$ does not change sign on the domain of integration.
Thus, from above,

$$h\tau = y'(s) - p(s) = \frac{y^{(4)}(\xi)}{3!} \underbrace{\boxed{\int_{t_n}^{t_{n+1}} (s - t_{n+1})(s - t_n)(s - t_{n-1})\ ds}}_{O(h^4)}$$

So dividing across by h we get the desired result. We set $\theta \in [\frac{1}{2}, 2]$ to control the scaling of the step size so that our method is stable.

1.2 Computing the Cholesky Factorization

We first 'construct' a matrix that has a valid Cholesky Factorization:

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 3 & 5 \\ 0 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 13 & 23 \\ 4 & 23 & 77 \end{bmatrix} =: A.$$

Now given the matrix A on the right, we want to factorize. We can write:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 13 & 23 \\ 4 & 23 & 77 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ b & d & 0 \\ c & e & f \end{bmatrix} \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix} = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 + d^2 & \\ ac & & \end{bmatrix}$$

If we want to specify an algorithm, we could perform gaussian elimination to perform this operation.

1.3 Error: Hermite Interpolation

We have

$$f(t) - p(t) = \frac{f^k(\xi)}{k!}[(t-t_1)_1^m \cdots (t-t_n)_n^m],$$

so the degree is $k = m_1 + \cdots + m_n$. Check that each of these three factors is inevitable.
Suppose:

$$f(t) = \sum_{k=-17}^{17} f_k e^{ikx},$$

where in order to perform this summation because it takes 35 operations. Evaluating the derivative is still relatively cheap:

$$f'(t) = \sum_{k=-17}^{17} f_k (ik) e^{ikt}$$

Hermite interpolation can be especially cheap on a mesh with equispaced points. If we are doing this via an FFT (forward), then this is especially cheap.
Additionally, we can evaluate the function at particular points as well as its derivatives at those points. Storing these into a table, we can just do a table lookup to eliminate the computation cost for calculating the derivatives at desired Hermite interpolation points.

2 Iterative Improvement

With a quarter of lecture's time available left, recall that we found:

$$||x||_p = \begin{cases} \sum_j |x_j|, & p = 1 \\ \sqrt{\sum_j x_j^2}, & p = 2 \\ \max |x_j|, & p = \infty \end{cases}$$
$$= \left(\sum_{j=1}^{n} |x_j|^p \right)^{1/p}$$

and additionally, for the norm of a matrix:

$$||A||_p = \begin{cases} \max_j \sum_i |a_{ij}|, & p = 1 \\ \lambda \max(A^T A), & p = 2 \\ \max_i \sum_j |a_{ij}|, & p = \infty \end{cases}$$

where our memory trick (mnemonic?) is that 1 is standing up and hence is the column sums.
Also, we took:

$$||A|| = \max_{||x||=1} ||Ax|| = \max_{x \neq 0} \frac{||Ax||}{||x||}$$

and this implies:

$$||Ax|| \leq ||A||\ ||x||$$
$$||AB|| \leq ||A||\ ||B||,$$

where

$$||A||\ ||A^{-1}|| \geq ||AA^{-1}|| = ||I|| = 1,$$

and we defined the condition number (not component-wise relative):

$$\kappa(A) = ||A||||A^{-1}|| \geq 1,$$

which Matlab recognizes as `rcond(A)`, where

$$\frac{1}{\text{rcond}(A)} = \infty$$

if A is not invertible. Also, we looked at the eigenvalues of A:

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

which gives $|\lambda| \leq ||A||$. We proved that

$$||A|| < 1 \implies (I - A) \text{ invertible}$$

We did this via geometric series; however, we want to be able to readily prove this without the help of such. Suppose $(I - A)x = 0$ and $x = Ax$, and so

$$||x|| = ||Ax|| \leq ||A||\ ||x||$$

(Intuitively, if $||A||$ is less than 0, then $||x||$ has to be zero to make this greater than.)

Proof.

$$\underbrace{(1 - ||A||)}_{>0} \underbrace{||x||}_{\geq 0} \leq 0$$

implies

$$0 \leq ||x|| \leq 0$$

□

This proves that the space of invertible matrices is an open set (open ball) centered around the identity. We ponder: Is A^{-1} a continuous function of A?
We check:

$$A^{-1} - B^{-1} = O(A - B)$$
$$A^{-1}\left(I - AB^{-1}\right) = A^{-1}(B - A)B^{-1},$$

and another useful fact (relative perturbation in A^{-1}):

$$\frac{||A^{-1} - B^{-1}||}{||A^{-1}||} \leq ||A|| \frac{||B - A||\ ||B^{-1}||}{||A||} = ||A||\ ||B^{-1}|| = \underbrace{||A||\ ||B^{-1}||}\ \frac{||B - A||}{||A||}$$

The norm of the product is less than or equal to the product of the norms (Cauchy Schwarz), so we got this result (of course assuming A, B invertible):

$$\boxed{\frac{||A^{-1} - B^{-1}||}{||A^{-1}||} \leq ||A|| \frac{||B - A||}{||A||} ||B^{-1}||}$$

The reason we went through all this is that we want to derive a way of solving $Ax = b$, where we **don't know** how to, and it's too large and difficult to try doing so. Instead, we use

$$\hat{A}\hat{x} = b,$$

where:
(1) $\hat{A}^{-1}$ is close to A^{-1}, or
(2) $\hat{A}$ is close to A,
and most importantly: $\hat{A}\hat{x} = b$ is **easy to solve!**
We write:

$$A = \begin{bmatrix} a_{11} & & U \\ & \ddots & \\ L & & \end{bmatrix} = D + L + U,$$

where we separate the diagonal, lower, and upper triangular parts of A. We can take, for instance:

$$\hat{A} := D + L$$

or

$$\hat{A} := D + U$$

we suspect this would be a good choice is our matrix A is **diagonally dominant.**

2.1 Generalizing:

Suppose $Ax = b, \hat{A}\hat{x} = b$. Compute $\hat{e} : \hat{x} + \hat{e}$ closer to x. We look at the residual (how much our computed solution fails to solve the exact equation):

$$b - A\hat{x} =: r := \text{residual of } \hat{x} \text{ in } A\hat{x} = b,$$

and so we define the error:

$$e := x - \hat{x},$$

so that

$$Ae = Ax - A\hat{x} = b - Ax = r,$$

and the residual r is computable, and so we take:

$$\hat{x} + e = x.$$

We get the exact solution at one step, but we don't know how to solve $A\hat{e} = r$, so we solve:

$$\hat{A}\hat{e} = r,$$

which gets us **closer** to the exact solution. It's cheap and gets us something better (to iterate on). $\hat{x} + \hat{e}$ is closer to x.

3 Office Hours

Let's do the local truncation error of 2019, 4A:

$$\begin{aligned} u_{n+1} &= u_n + \frac{h}{2} f\left(\frac{1}{3}u_n + \frac{2}{3}u_{n+1}\right) + \frac{h}{2} f\left(\frac{2}{3}u_n + \frac{1}{3}u_{n+1}\right) \\ &= u_n + \frac{h}{2}k_1 + \frac{h}{2}k_2, \end{aligned}$$

and so we have:

$$\begin{aligned} k_1 &= f(\frac{1}{3}u_n + \frac{2}{3}\left(u_n + \frac{h}{2}k_1 + \frac{h}{2}k_2)\right) \\ &= f\left(u_n + \frac{1}{3}hk_1 + h\frac{1}{3}k_2\right) \end{aligned}$$

Because $b_1 + b_2 = 1$, so $O(h)$ is satisfied. The second order condition was

$$b^T Ae = \frac{1}{2},$$

so we check :

$$\frac{1}{2}e^T Ae = \frac{1}{2},$$

so it checks out, and we have second-order accuracy. The left entries c are the row sums of the matrix A.
Now to do the local truncation error, we had:

$$u_{n+1} = u_n + h\left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right),$$

so we have the local truncation error:

$$\begin{aligned}\tau &= \frac{y_{n+1}-y_n}{h} - \frac{1}{2}k_1 - \frac{1}{2}k_2 \\ &= \frac{\cancel{y} + hy' + \frac{1}{2}h^2y'' - \cancel{y}}{h} + O(h^2) - \frac{1}{2}k_1 - \frac{1}{2}k_2 \\ &= y' + \frac{1}{2}hy'' - \frac{1}{2}\left(k_1+k_2\right) + O(h^2) \\ &= f + \frac{1}{2}hf'(y)f(y) - (\cdots),\end{aligned}$$

where

$$y' = f(y), \qquad y'' = f'(y)f(y)$$

and taking $g = k_1 + k_2 = f(y) + O(h)$,

$$\begin{aligned}g = k_1 + k_2 &= f\left(y + \frac{h}{3}(k_1+k_2)\right) + f\left(y + \frac{h}{6}(k_1+k_2)\right) \\ &= f\left(y + \frac{h}{3}g\right) + f\left(y + \frac{h}{6}g\right) \\ &= f(y) + f'(y)\frac{h}{3}g + f(y) + \frac{h}{6}g + O(h^2) \\ &= 2f(y) + f'(y)\frac{h}{2}g + O(h^2),\end{aligned}$$

but Strain gets stuck here and says this may not be second-order accurate (as the problem would suggest), so we made a mistake (the problem is correct)! We look back into our notes, and Strain newly derives:

Second-order (accuracy) condition:

$$\sum b_i \sum a_{ij} = \frac{1}{2}.$$

Strain goes back to notice that as $h \to 0$, we have $g \to 2f$, so we found (and fixed) our issue.
Now for the second part of the problem,

$$\begin{aligned}f(y) &= \lambda y \\ u_{n+1} &= u_n + \frac{h}{2}f\left(\frac{1}{3}u_n + \frac{2}{3}u_{n+1}\right) + \frac{h}{2}f\left(\frac{2}{3}u_n + \frac{1}{3}u_{n+1}\right) \\ &= u_n + \frac{h\lambda}{6}u_n + \frac{h\lambda}{3}u_n + \frac{h\lambda}{3}u_{n+1} + \frac{h\lambda}{6}u_{n+1} \\ &= \left(1 + \frac{h\lambda}{2}\right)u_n + \frac{h\lambda}{2}u_{n+1} \\ \left(1 - \frac{h\lambda}{2}\right)u_{n+1} &= \left(1 + \frac{h\lambda}{2}\right)u_n,\end{aligned}$$

which gives the desired result:

$$u_{n+1} = \frac{1 + \lambda h/2}{1 - \lambda h/2}u_n$$

The other way that we found in lecture to do this is via the Butcher array:

$$k = \lambda(u_n e + hAk)$$

so we have:

$$\begin{aligned}
(I - h\lambda A)\, k &= \lambda u_n e \\
k &= (I - h\lambda A)^{-1}\, \lambda u_n e \\
u_{n+1} &= u_n + hb^T k \\
&= u_n + hb^T (I - h\lambda A)^{-1} \lambda u_k e \\
&= \left(1 + h\lambda b^T (I - h\lambda A)^{-1} e\right) u_n
\end{aligned}$$

and hence we write:

$$R(z) = 1 + z \underbrace{b^T (I - zA)^{-1} e}_{b^T e + z b^T A e + z^2 b^T A^2 e} .$$

But we conclude that it's better to do it the first way, algebrically on a timed exam.

Math 128A, Summer 2019
Lecture 31, Wednesday 8/14/2019

CLASS ANNOUNCEMENTS: Final exam tomorrow.

1 Review: Final Exam Spring 2019

1.1 Problem 5B

We open with a classmate asking about the following problem:

Problem 5B. Suppose A is a diagonally dominant $n \times n$ matrx, and a lower-triangular matrix L with diagonal entries equal to 1 and an invertible upper triangular matrix U satisfy:

$$LU = A + F$$

for some $n \times n$ matrix F.

(1) Find a lower-triangular matrix B with zero diagonal entries and an upper triangular matrix C such that

$$L(I + B)(I + C)U = A + L\ B\ C\ U.$$

(2) Let $\hat{L} := L(I + B)$ and $\hat{U} := (I + C)U$. Show that the residual

$$\hat{L}\hat{U} - A = O(||F||^2)$$

as $||F|| \to 0$.

Strain opens with a blind shot, eventually getting stuck without using $LU = A + F$ given in the problem. We'll call this 'background' as it is helpful intuition.

1.2 Background

Recall that when we solve for the Cholesky factorization, we have many different ways to do this.

$$R^T R = A \implies (R + E)^T (R + E) - A = E^T E$$

In this particular case, we do a weird thing, which is to take the quadratic piece of the LHS and put it on the LHS. We aren't solving for the quadratic anymore but we're solving something more interesting. We're solving for the residual. If the residual is small, then the solution gets more interesting.
One way to solve $LU = A$ is to look for corrections such that:

$$(L + \Delta L)(U + \Delta U) = A$$

which will satisfy a coupled quadratic equation (which would be a pain), so we do:

$$(L + \Delta L)(U + \Delta U) = A + \underbrace{\Delta L \cdot \Delta U}$$

via group theory. The residual in the corrected equation which we are trying to solve will be $\Delta L \cdot \Delta U$. However, it is more natural to (instead) take:

$$L(I + \Delta L)(I + \Delta U)U = A$$

Recall that in the related lecture, we had a bunch of 'funny looking stuff' (according to Strain). This was because we did the additive version instead of the multiplicative version (as given this problem).
Now we take (to get a linear equation):

$$L(I + \Delta L)(I + \Delta U)U = A + L \cdot \Delta L \cdot \Delta U \cdot U$$

which gives us an equation to solve:

$$L(\Delta L + \Delta U)U = A - LU$$

Multiplying across, we get:

$$\Delta L + \Delta U = L^{-1}AU^{-1} - I$$

It turns out this gives n^2 equations and n^2 unknowns. The two matrices ΔL and ΔU are orthoginal in that ΔL is lower triangular and ΔU is upper triangular. Because they are orthogonal, this suggests that we should mention the Frobenius norm again:

$$||A||_F^2$$

Because we talked about the great things about operator norms before, we'll now talk about the great things about the Frobenius norm.

$$||A||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2,$$

and another way to think about this is $\langle A, A\rangle$, but notice this is NOT the same thing as A^TA. Really what's happening is that we're taking column sums of squared elements, and summing the rows. So we have:

$$||A||_F^2 = \langle A, A\rangle = \sum_{i=1}^{n} \left(\begin{bmatrix} \cdots & \cdots & \cdots \\ & & \end{bmatrix} \begin{bmatrix} \vdots & \\ \vdots & \\ \vdots & \end{bmatrix} \right)_{ii} = \mathrm{tr}(A^TA),$$

the trace of A^TA. Generally, a good way to define the inner product of two matrices ss :

$$\langle A, B\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ij} = \mathrm{tr}(A^TB).$$

We mention this because in our main problem,

$$\langle \Delta L, \Delta U\rangle = 0.$$

Strain checks again the hypothesis and given information in the problem and starts again.

1.3 Solution.

In our problem, we had:

$$LU = A + F,$$

so

$$L(I+\Delta L)(I+\Delta U)U - A = L\cdot\Delta L\cdot\Delta U\cdot U,$$

where the LHS gives us the residual. So this gives:

$$\begin{aligned} LU + L(\Delta L+\Delta U)U - A &= 0 \\ L(\Delta L+\Delta U)U &= A - LU = -F \\ \Delta L &= \text{slt}(-L^{-1}FU^{-1}) \\ \Delta U &= \text{upt}(-L^{-1}), \end{aligned}$$

so we have:

$$L\cdot\Delta L\cdot\Delta U\cdot U = L\text{slt}(-L^{-1}FU^{-1})\text{upt}(-L^{-1}FU^{-1})U$$

These matrices slt, upt (strictly lower triangular, upper triangular) are just projections to lower or upper triangular matrices. In an inner product space, projections always decrease the norm. The question is if:

$$||AB||_F \overset{?}{\leq} ||A||_F\ ||B||_F,$$

where we recall $||I||_F = \sqrt{n}$.
It turns out that we need simply need to change one of the norms on the RHS to the 2-norm. That is,

$$||AB||_F \leq ||A||_2||B||_F.$$

To see this, consider:

$$||AB||_F^2 = ||Ab_1||_2^2 + \cdots + ||Ab_n||_2^2 \leq ||A||_2^2||B||_F^2.$$

So in our original problem, we have that the residual:

$$\text{residual} = R = \hat{L}\hat{U} - A$$

is bounded by:

$$||\hat{L}\hat{U} - A|| \leq \underbrace{\kappa(L)\kappa(U)}\ ||F||^2\ \underbrace{||U^{-1}||||L^{-1}||} = O(||F||^2),$$

where all the other factors are all constants unrelated to $||F||$, so this is our required result (but doesn't look pretty).

2 Symmetric Arrowhead Matrices

Now from Spring 2018 Final Exam (as Strain mentions is stolen from Professor Gu), Let

$$A := \begin{bmatrix} d_1 & 0 & \cdots & 0 & r_1 \\ 0 & d_2 & 0 & \cdots & r_2 \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & d_{n-1} & r_{n-1} \\ r_1 & r_2 & \cdots & r_{n-1} & d_n \end{bmatrix} = D + re_n^T + e_nr^T,$$

where $e_n^Tr = 0$, be a symmetric arrowhead matrix. Develop an algorithm for computing the Cholesky factorization fo A in $O(n)$ scalar floating point operations. Use your algorithm to find a condition on the diagonal matrix D and the vector r which determines when the matrix A is positive definite.

2.1 Solution.

We call this an arrowhead matrix. We don't want to perform pivoting because that would destroy the nice structure of this matrix. There are a few good ideas, but our best idea here is to try Cholesky factorization.
The question is if in the Cholesky factorization, we have enough degrees of freedom to solve what we need. It turns out that we have enough zeros to 'move' via Gaussian elimination. Our condition on the diagonal is something like:

$$a_{nn} - a_{n1} - a_{n2} - \cdots > 0$$

2.2 Algorithm for Computing the Cholesky Factorization of A in $O(n)$

The algorithm would be to take the diagonal entry and 'kill' the last entry in the row. We take:

$$A = D + re_n^T + e_n r^T e_n^T r = 0$$

and we think about a factorization of this. We guess and check Λ to be like D:

$$R = \Lambda + \rho e_n^T e_n^T \rho = 0$$

so,

$$\begin{aligned} R^T R &= (\Lambda + \rho e_n^T)^T (\Lambda + \rho e_n^T) \\ &= (\Lambda + e_n \rho^T)(\Lambda + \rho e_n^T) \\ &= \Lambda^2 + e_n \rho^T \rho e_n^T + e_n \rho^T \Lambda + \Lambda \rho e_n^T \end{aligned}$$

Strain reads off the result:

$$\Lambda^2 + \rho^T \rho e_n e_n^T = D, \quad r = \Lambda \rho,$$

but we actually have the following for free:

$$\boxed{\lambda_1^2 = d_1 \cdots \lambda_{n-1}^2 = d_{n-1}},$$

and likewise onwards, but the only one we don't know is:

$$\boxed{\lambda_n^2 = d_n - \sum_{j=1}^{n-1} \frac{r_j^2}{d_j}}.$$

To get ρ, we use $r = \Lambda\rho \implies \rho = \Lambda^{-1} r$. Our algorithm is actually just to write down the exact expression and evaluate!

```
for i = 1:n-1
  R_ii = sqrt(d_i)

R_nn = sqrt( d_n - sum_{j=1}^{n-1} (r_j)^2/d_j )

for (i = 1:n-1)
  R_in = r_i / R_ii
```

The condition for our matrix to be positive definite is:

$$\partial_n - \sum_{j=1}^{n-1} \frac{r_j^2}{d_j} > 0$$

3 Numerical Integration Rules

We want a numerical integration rule where we characterize good points and weights for:

$$\int_0^1 f(t)\ dt = \sum_{j=1}^{n} w_i f(t_i)$$

What's going on here is that the quality of our quadrature rule is to separate these two things.
1) $|f(t) - p(t)| \leq \epsilon$ Find a class of polynomials that achieves this bound for a given f.

$$\begin{aligned}\int_0^1 f(t)\ dt &= \int_0^1 f(t) - p(t)\ dt + \underbrace{\int_0^1 p(t)\ dt} \\ &= \int_0^1 \left(f(t) - p(t)\right)\ dt + \sum_{i=1}^{n} w_i \left(p(t_i) - f(t_i) + f(t_i)\right)\end{aligned}$$

so we move a term over and use an infinite number of triangle inequalities to get:

$$\begin{aligned}\left|\int_0^1 f(t)\ dt - \sum_{i=1}^{n} w_i f(t_i)\right| &\leq \int_0^1 |f(t) - p(t)|\ dt + \sum_{i=1}^{n} |w_i| \underbrace{|p(t_i) - f(t_i)|}_{\epsilon} \\ &\leq \underbrace{\left(1 + \sum_{i=1}^{n} |w_i|\right)}_{\text{cond. num for quad rule}} \epsilon\end{aligned}$$

As an aside, we can get the total sum of the weights by the condition set upon the integration rule. In our present case we usually test $f = 1$ to get weight sums of 1 or 2 because our quadrature rule makes integration exact for polynomials of degree less than or equal to n. That is,

$$0 < \int_{-1}^{1} L_j(t)^2\ dt = \sum_{i=1}^{n} L_j(t_i)^2 w_i = w_j$$

3.1 Example: Integration Weights for Exponentials

Suppose $\lambda_j > 0$ for $1 \leq j \leq n$, so:

$$\frac{1}{\lambda_j} = \int_0^{\infty} e^{-\lambda_j x}\ dx = \int_{i=1}^{n} e^{\lambda_j x_i} w_i,$$

which gives an $n \times n$ linear system we can solve for weights. Doing so gives:

$$\int_0^{\infty} f(x)\ dx = \int_{i=1}^{n} w_i f(x_i),$$

whenever $f(x)$ is some linear combination of decaying exponentials:

$$f(x) = \sum_{i=1}^{n} f_j e^{-\lambda_j x}$$

3.2 Example: ODE, Linear Stability

Let's say we're solving some explicit scheme (so we could see how bad it is).

$$u_{n+1} = u_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right),$$

and consider $f(y) = \lambda y$. Then:

$$u_{n+1} = u_n + h\lambda\left(\frac{3}{2}u_n - \frac{1}{2}u_{n-1}\right)$$
$$u_{n+1} = \left(1 + \frac{3}{2}h\lambda\right)u_n - \frac{1}{2}h\lambda u_{n-1},$$

and this looks like a 2-term recurrence relation. Let $z := h\lambda$, and we check the characteristic roots:

$$r^2 - \left(1 + \frac{3}{2}z\right)r + \frac{1}{2}z = 0,$$

and we have a problem (we don't need to look any further). If we have some r_1, r_2 roots satisfying this,

$$(r - r_1)(r - r_2) = 0$$

then

$$r_1 r_2 = \frac{1}{2}z,$$

and z will vary and explode if one of the roots is outside the unit circle. This two-step explicit Adams (Bashforth) would not be good for stiff equations.

4 Iterative Refinement

The compute bound is that multiplying matrices (inputting n^2 inputs costs n^3 time), but numerical linear algebraists.
(This got interrupted to answer more questions above about past exam problems and related hypotheticals).

Lecture ends here.

This was the last lecture; good luck on the final exam (lecture/class 32), and have a nice summer!