Introduction to Numerical Analysis

MATH UA-252-005 / MA-UY.4424-C

Spring 2022

Chapter 1

Preliminary material: Python and LaTeX

For your homework in this course, you'll need to use Python for computational exercises and LaTeX for written exercises.

1.1 Programming in computational science

Python is a high-level programming language which is used in some capacity in nearly every field of software and computing. It isn't the best programming language, and it isn't the worst—but it is very popular. This and the fact that it has a simple and readable syntax make it a fine choice for a first language if you have never programmed before.

In computational science, there are several other popular high-level programming languages: MATLAB, Julia, and R.

MATLAB is older and more well-established. It is a closed source programming language and computing environment, produced and sold by The MathWorks. The style of Python programming that you'll learn in this class can be easily translated to MATLAB. Because MATLAB is developed by a large number of software engineers who are paid to do so, it is more stable, more consistent, and tends to have much better documentation than the alternatives. A stated goal of The MathWorks is to ensure that MATLAB code is backwards compatible—that is, The MathWorks tries to make sure that new versions of MATLAB don't break old code that works (this is not the case for Python!). This is a useful for feature for people who don't have time to spend modernizing old code. On the other hand, it means old programming language features which turned out to be a poor idea aren't removed from the language. For this reason, MATLAB code sometimes feels a little outdated or awkward.

[SFP: Give a rundown of Julia and R.]

1.2 Programming assignments and Python

The first thing to do is get access to Python. Installing Python will be a little different depending on the operating system. Being able to troubleshoot your own problems is a key skill to develop when

doing any kind of programming. For this reason, instead of giving elaborate instructions for how to set up Python, please make an attempt to figure it out on your own. Start by visiting python.org and following the instructions there for your platform.

One caveat: Python is not particularly well supported on Windows. To use Python on Windows, it is highly recommended to either use a virtual machine to host Linux or Windows Subsystem for Linux (WSL). To use a virtual machine, install VirtualBox and some flavor of Linux, such as Ubuntu or Fedora.

Regardless of which operating system you're running, once you have Python installed you should be able to open a terminal and type `python`. You should see something like this:

```
Python 3.9.10 (main, Jan 15 2022, 12:21:28)
[Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> ■
```

This is the Python prompt. It may look a little different depending on your setup. This is for Python running on macOS (hence, "`darwin`").

Since Python doesn't enforce compatibility between different versions, we have to take action to ensure that code that runs for one person will run for whoever is grading their submission. Install at least Python 3.8 or 3.9. The current version is Python 3.10. For our purposes, the differences between these versions shouldn't be too great.

Additionally, Python provides a mechanism called a virtual environment, which is used for setting up an environment which a particular set of installed packages with fixed version numbers in case different versions of a particular package are needed for different projects developed on a single computer.

For the programming assignments in this class, we will all use the same, minimalist virtual environment which will give us access to Python (which itself already has a huge amount of inbuilt functionality), and three libraries which give Python functionality roughly equivalent to MATLAB: numpy, scipy, and matplotlib. We also may use sympy occasionally. These libraries will be introduced throughout the course. We will also use an enhanced Python REPL called IPython.

Virtual environments can be built from "requirements files". These are typically files named `requirements.txt` containing a list of packages along with their versions. To set up a virtual environment to use for your programming assignments, do the following:

1. Open up a terminal and create a directory somewhere you would like to keep your programming assignments, e.g. "`hw`"; then, `cd` into that directory, and use Python to create a new virtual environment:

   ```
   % mkdir hw
   % cd hw
   % python -m venv .
   ```

Here, "." refers to the current directory. If you aren't familiar with how to use the shell, please do a bit of background reading and get familiar with it. Googling "`shell intro`" should turn up plenty of resources (one such example).

2. Activate the virtual environment by running:

```
% source bin/activate
```

To shed some light on this mysterious command: the `source` command is a shell built-in which evaluates shell commands from a file; so, `bin/activate` is just the path to a file containing shell commands which are run to make the virtual environment active. If you're curious, you can run `cat bin/activate` to see what they are.

3. Copy the following lines into a new file named `requirements.txt` in the directory which you've just created:

```
ipython==8.0.1
matplotlib==3.5.1
numpy==1.22.1
scipy==1.7.3
sympy==1.92
```

4. Use pip to install these packages:

```
% pip install -r requirements.txt
```

5. Finally, verify that we installed everything successfully and that it works. First, run `ipython` to start IPython:

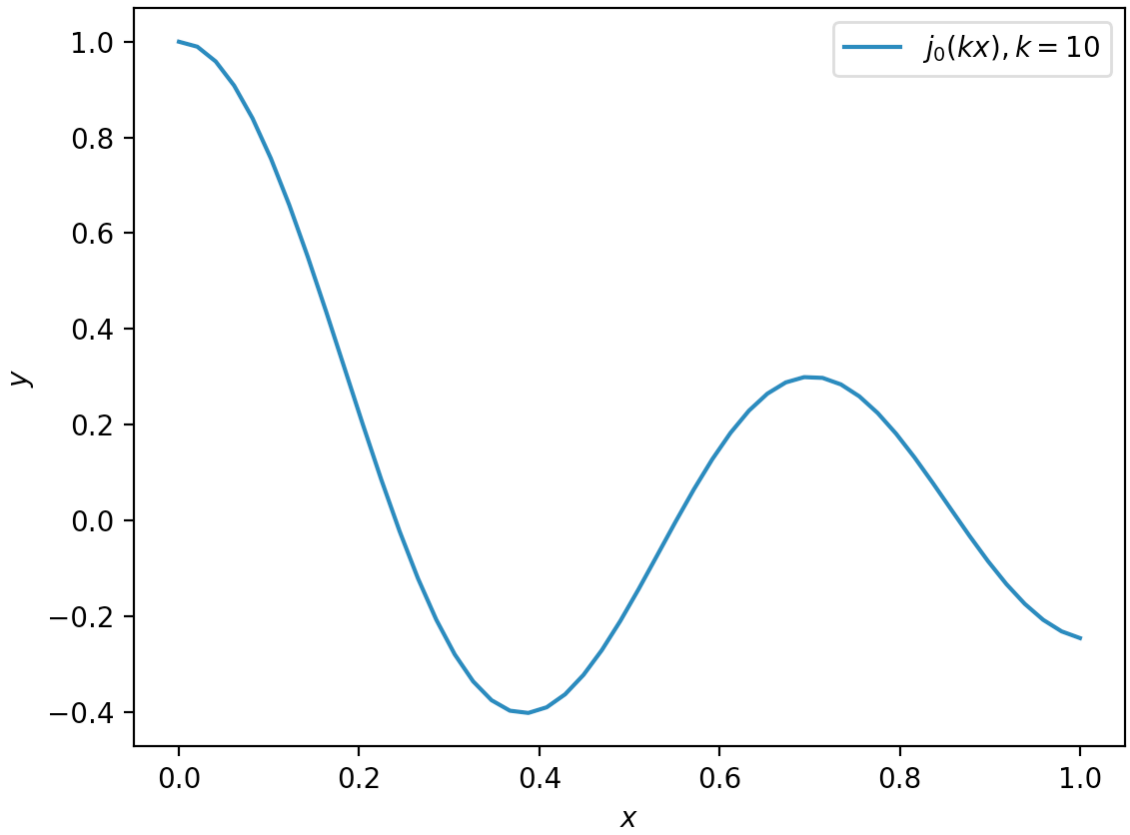
```
% ipython
Python 3.9.10 (main, Jan 15 2022, 12:21:28)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: ■
```

Next, we'll import numpy, scipy, and matplotlib do a few simple things to make sure they're operational.

```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: from scipy.special import j0

In [4]: x = np.linspace(0, 1)

In [5]: k = 10

In [6]: y = j0(k * x)

In [7]: plt.figure()
Out[7]: <Figure size 1280x960 with 0 Axes>

In [8]: plt.plot(x, y, label=r"$j_0(kx), k = 10$")
Out[8]: [<matplotlib.lines.Line2D at 0x12d2037f0>]

In [9]: plt.legend()
Out[9]: <matplotlib.legend.Legend at 0x12d1f56d0>

In [10]: plt.xlabel("$x$")
Out[10]: Text(0.5, 0, '$x$')

In [11]: plt.ylabel("$y$")
Out[11]: Text(0, 0.5, '$y$')

In [12]: plt.show()
```
■

Please type each of these lines in yourself! This shows a fairly simple, fairly typical IPython + numpy + scipy + matplotlib session. You could do something very similar in MATLAB or Julia if you were so inclined (give it a try if you're interested). After the final line containing `plt.show()` is executed, you should see something like this:

double



root

In the last step, I'm leaving each of the Python commands unexplained. What they do should be obvious from the context. It's your job to do some research to look these invocations up to understand them a bit more concretely.

Some LaTeX was also slipped into the last step. Everything between a pair of dollar signs ($) is some LaTeX in "math mode"—the form of LaTeX used to typeset mathematical formulae. In this case, it was used to typeset the lone entry in the plot's legend and the plot's x- and y-axes.

The function typeset is the zeroth order Bessel function of the first kind. This is an important special function that comes up frequently in mathematical physics. It is defined to be the solution of the ordinary differential equation (ODE) of the form:

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - n^2)y = 0 \tag{1.1}$$

for $n = 0$ which is nonsingular at the origin.

Chapter 2

Fixed point iterations

[SFP: Scribed by Riya Makoshi.]

2.0.1 The Babylonian Method

At the end of the last class we saw the Babylonian Algorithm, an iterative method for computing square roots by hand.

$$y_{k+1} = \frac{1}{2}\left(\frac{x}{y_k} + y_k\right) \tag{2.1}$$

We also learned that if $y_0 > 0$ (the initial iterate), then y_k converges to $\sqrt{x}$. With each additional iteration, the number of digits of accuracy also doubles and eventually plateaus. This ties into the intricacies of floating point arithmetic where double precision floats roughly have the capacity for 16 digits of agreement (capped at a fourth iteration of the Babylonian Algorithm)

The Babylonian Method is an example of a fixed-point iteration which is a term that describes methods used to compute fixed points of functions.

2.0.2 FPI - Definition

Fixed Point Iteration 1. Let $f : \mathbb{R} \to \mathbb{R}_1$ and let x_0 be fixed (somehow) then we'll call the sequence generated by $x_{k+1} = f(x_k)$ where $(k > 0)$ a simple iteration (aka a fixed point iteration).

2.0.3 Fixed Point Iteration Conditions

Before looking further there are a few things we need to think about.

1. First, is there anything special about fixed points?
2. Second, if we have an iteration like $x_{k+1} = f(x_k)$ should we automatically expect convergence to a fixed point?

As a side note, in class we learned that Numerical Analysis largely involves designing algorithms to solve continuous problems by finding approximate solutions. Often, if the solution for a problem exists at all and the solution is unique, we will be more likely to succeed in our endeavors. Alternatively, Computer Science is more often concerned with finding the exact solution to a discrete problem.

In general Numerical Analysis is concerned with two main concepts:

1. Existence
2. Uniqueness

To apply these concepts to fixed point iterations we can ask - for a function f should a fixed point exist? If it does, is it unique?

To take things further, the Babylonian Algorithm shows us:

$$y_{k+1} = f(y_k)$$
$$f(y) = \frac{1}{2}\left(\frac{x}{y} + y\right)$$

This begs the question, does $(y_k)_{k=0}^{\infty}$ converge?

Cauchy Sequences 1. $(y_k)_{k=0}^{\infty}$ is Cauchy (converges) if the following conditions are met:

$$\forall \epsilon > 0 \exists N(\epsilon) \geq 0 \tag{2.2}$$

$$\forall m, n \geq N(\epsilon) \,:\, D(y_m, y_n) < \epsilon \tag{2.3}$$

- N = Some integer
- ϵ = Error tolerance
- D = Distance

2.0.4 Determining the Existence of Fixed Points

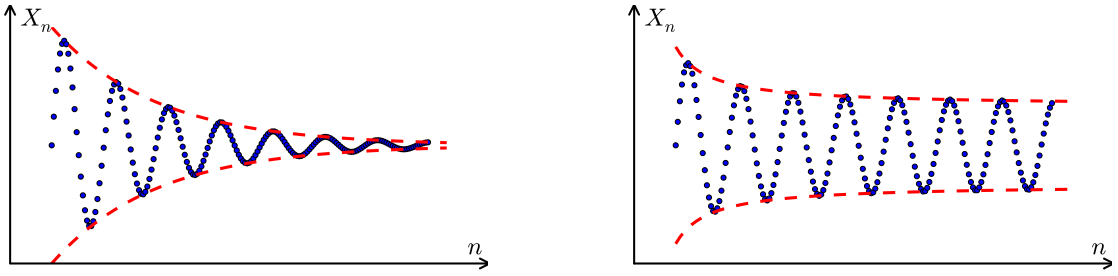
Intermediate Value Theorem

The first way to determine the existence of fixed points is to utilize the Intermediate Value theorem or IVT.

Intermediate Value Theorem 1. The theorem states that if f is real-valued: $f \in C^0([a, b])$ continuous on the interval a,b where a,b is closed and bounded. Then for each y such that

$$\min_{a \leq x \leq b} f(x) \leq y \leq \max_{a \leq x \leq b} f(x) \tag{2.4}$$

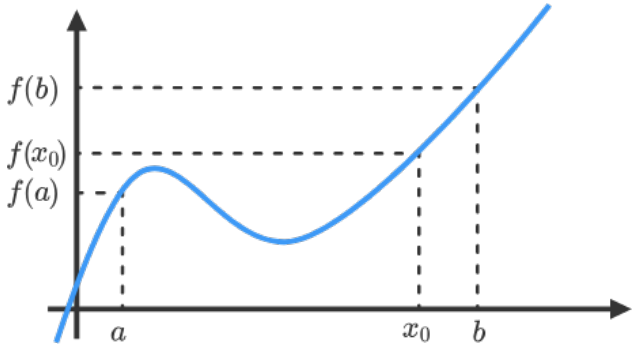
there exists $x \in [a, b]$ such that $y = f(x)$



(a) Cauchy sequence

(b) Non-Cauchy sequence

Figure 2.1: As seen in the figures above, a Cauchy sequence converges to an "ultimate destination", or in other words, a limit clearly exists

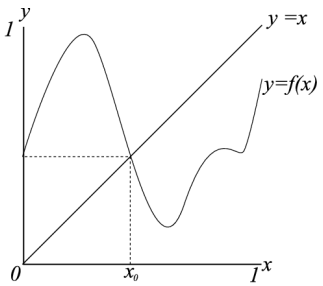


(a) Intermediate Value Theorem

Figure 2.2: The Intermediate Value Theorem establishes the existence any given value between $f(a)$ and $f(b)$ at some point within $[a, b]$ given that the function in continuous throughout the interval.

Brouwer's Fixed Point Theorem (in 1D)

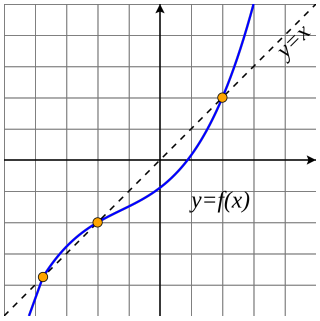
Brouwer's Fixed Point Theorem 1. Assume that f is real-valued and assume that $f([a, b]) \leq [a, b]$. Then $\exists \xi$ such that $f(\xi) = \xi$



(a) Brouwer's Theorem

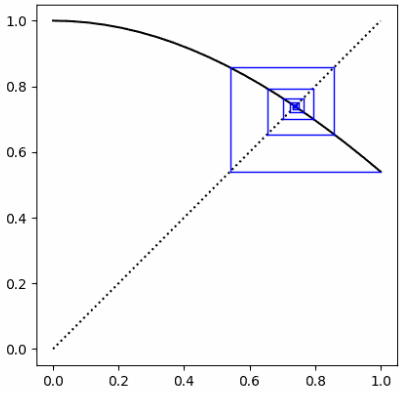
A fixed point for $f : \mathbb{R} \to \mathbb{R}$ is a point where $y = x$ intersects the graph of f. However, it is

not always guaranteed that a fixed point will exist or that there will be only one fixed point.



(a) Multiple Fixed Points

As seen above, a function f can have multiple fixed points. If we have identified multiple points, how can we tell which fixed point the iteration $x_{k+1} = f(x_k)$. Food for thought: We can utilize a mapping technique called cobweb plotting to graphically solve for a fixed point on f. However, depending on which part of the function you start, the cobweb plot will reach a different fixed point given that there are multiple fixed points.



(a) Cobweb Plot

Figure 2.5: The cobweb plot reaches a fixed point by following a path back and forth across $f \subseteq y = x$.

Note: If f is continuous and $f([a, b]) \leq [a, b]$ and if x_k generated by $x_{k+1} = f(x_k)$ converges to ξ then:

$$\xi = \lim_{k\to\infty} x_{k+1} = \lim_{k\to\infty} f(x_k) = f(\lim_{k\to\infty} x_k) = f(\xi). \tag{2.5}$$

This tells us that if the preconditions for Brouwer's Theorem hold and the sequence converges, then it will converge to a fixed point. Given this, we are left with the questions - Does x_k converge and to which fixed point will it converge to?

2.0.5 Contractions and Contraction Mapping

To determine a unique fixed-point of convergence it is viable to use contractions or contraction mapping.

Contraction Definition and Lipschitz Constants

Contraction Definition 1. Let $f \leftarrow C^0[a,b]$ on a finite interval $[a,b]$. Then f is a contraction if $\exists L < 1$ such that $|f(x) - f(y)| < L|x - y|$ for all x, y.

L is called a Lipschitz Constant for f over the interval $[a,b]$. To define the term further: If $L > \max_{a \le x \le b} |f'(x)|$ then L is a Lipschitz constant for f over $[a,b]$. Note: $L < 1$ is not required for a Lipschitz constant.

Contraction Mapping Theorem

Contraction Theorem 1. If $f \in C^0[a,b]$, $f([a,b]) \subseteq [a,b]$, and f is a contraction on $[a,b]$. Then f has a unique fixed point ξ on $[a,b]$ and $x_{k+1} = f(x_k)$ converges to ξ. To define further - let $x_{k+1} = f(x_k)$ generate a sequence and assume it converges to ξ. Then the basin of attraction of ξ is all x_0 such that $x_k \to \xi$.

To apply the contraction mapping theorem to f over $[a,b]$:

1. Check if $f \leftarrow C^0[a,b]$ if $f([a,b]) \le [a,b]$
2. Compute $L = max_{a \le x \le b} |f'(x)|$
3. Check if $L < 1$
4. If the above condition is met, there exists a unique fixed point for f, call it ξ in $[a,b]$.
5. Compute ξ: Pick any $x_0 \leftarrow [a,b]$ and run $x_{k+1} = f(x_k)$ until convergence.
    - At the fixed point $x_k = \xi$ and $x_{k+1} = f(x_k) = f(\xi) = \xi$
6. Compute $x_{k+1} - x_k$ to check accuracy

Note: If you are interested in further research: more general questions can be answered by discrete dynamical systems (Henon maps, chaotic iterated maps (2D), etc.).

2.1 Homework Notes

Sturm Chain Definition 1. Let p, where $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 ....$ Define $p_0 = p, p_1 = p'(x) = \frac{dp}{dx}$ For $n \ge 2$ define $q_n = [\frac{p_{n-2}}{p_{n-1}}]$ where q_n is the quotient result of polynomial long division (without the remainder). Then define, for the same n, $p_n = q_n(p_{n-1}) - p_{n-2}$ stopping when p_N is constant. The resulting sequence $p_0, ..., p_N$ is called a Sturm chain.

2.1.1 Sturm's Theorem

Sturm Chain Theorem 1. Let p be a polynomial. Let $a < b$ define a bounded interval $[a, b]$ and let $p_0, ..., p_N$ be the Sturm chain corresponding to p. Now consider the table below where Δ_a and Δ_b are the number of sign changes in each row.

$p_0 a$	$p_1 a$	...	$p_n a$	$\rightarrow$	Δ_a
$p_0 b$	$p_1 b$	...	$p_n b$	$\rightarrow$	Δ_b

The number of real roots in (a, b) is $|\Delta_b - \Delta_a|$.

Sturm's Theorem Example

For example, say you are given a polynomial $p_0 = x^5 - 3x - 1$. You can calculate the Sturm Chain below. Note that it took three iterations to reach a constant.

$$
\begin{aligned}
p_0 &= x^5 - 3x - 1 \\
p_1 &= 5x^4 - 3 \\
p_0 &= \frac{1}{5}(12x + 5) \\
p_3 &= \frac{59083}{20736}
\end{aligned}
$$

You can then use the Sturm chain to generate the following sign change table.

x	Sign p_0	Sign p_1	Sign p_2	Sign p_3	Δ
-2	$-$	$+$	$-$	$+$	3
0	$-$	$-$	$+$	$+$	1
2	$+$	$+$	$+$	$+$	0

This table can now be used to identify the number of roots in each interval. $\Delta_{x=0} - \Delta_{x=2} = 1$ therefore there is only one root in this interval. This is a good place to use a tool such as `brentq` to identify the single root. However, $\Delta_{x=-2} - \Delta_{x=0} = 2$ meaning there are multiple roots in this interval. To proceed, Sturm's theorem will need to be reapplied until you are able to separate the roots into separate intervals.

For the homework, a viable method is combining Sturm's theorem with binary search using bisection. Utilizing python generators could also be beneficial - link to the documentation can be found here.

Chapter 3

Solving nonlinear equations in 1D

Nikhil Isac January 31st 2022

3.1 Note on Programming Homework #1

We will be using Brentq, which is a 'hybrid rootfinder' which implements Brent's method.
Brentq solves: "find $x \in [a, b]$ such that $f(x) = 0$"
Signature: `x = brentq(f,a,b,tol = None)`

- `tol`: "tolerance," how accurately the root is found.
- `f`: An instance of something on Python which is callable
- If we use "$f(x)$", where x is a float, then $f(x)$ is a float
  `From scipy.optimize` we can `import brentq`
  `from ipython, type ?brentq` for docs or just google "brentq scipy"

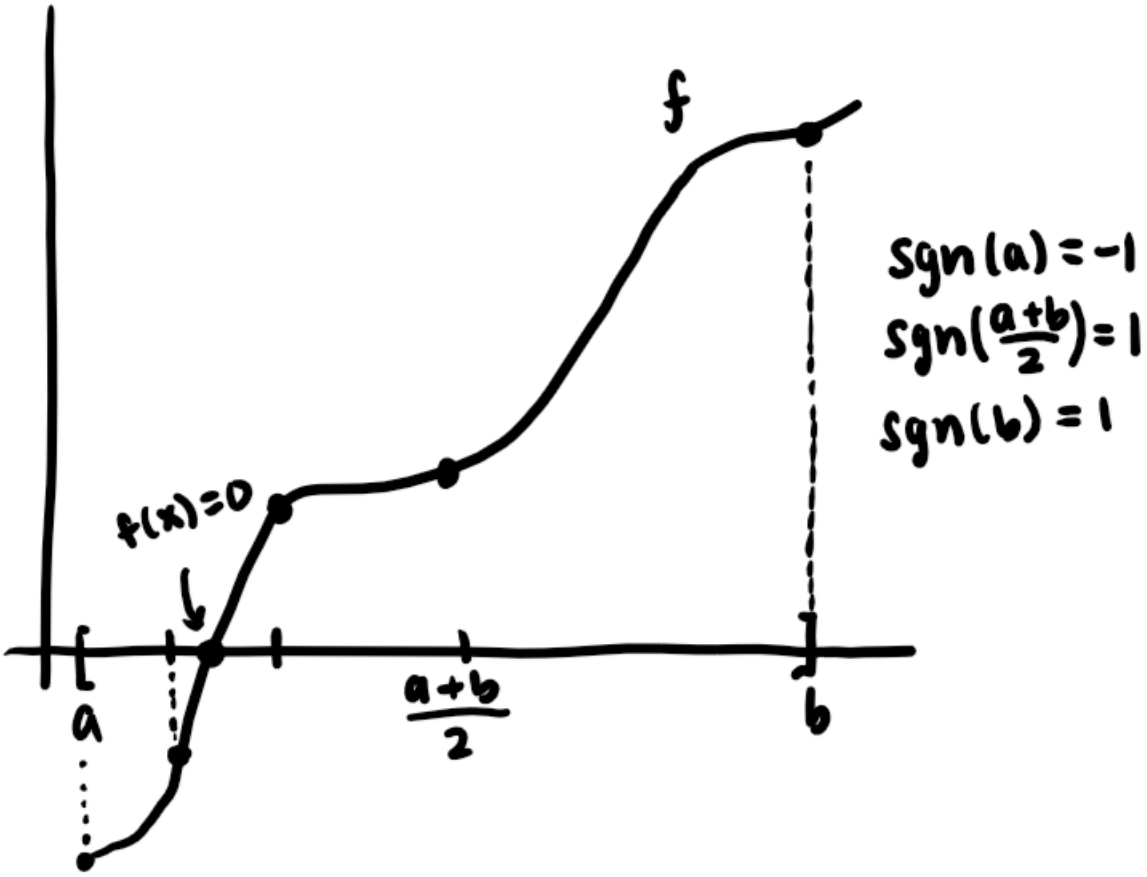
Question: $p(x) = (x - 10^{-16})(x + 10^{-16}) = x^2 - 10^{-32} \approx x^2$
Question: how accurate does findroots need to be?
Answer: `roots = findroots(p,a,b,tol)` (where tol > 0)

Def: an open set E in $\mathbb{R}$ is a subset of the real line such that if $x \in E$, then there exists some $\epsilon > 0$ such that $B_\epsilon = \xi y \in \mathbb{R} : |x - y| < \epsilon$

Def: a set F $\subseteq \mathbb{R}$ is closed if it's complement is open.
$F^c = \{y = \mathbb{R} : y \notin F\}$
e.g. Is the set $[1, \infty)$ closed?
well $[1, \infty)^c = (-\infty, 1)$... open
$\Rightarrow [1, \infty)$ is closed.
Note: For the contraction mapping theorem, the assumption on the interval $[a, b]$ is that it is closed AND bounded or compact

3.2 Iterative Solution of Equations

(or in 1D . . . “rootfinding".)
We want to solve $f(x) = 0$ for $f : \mathbb{R} \to \mathbb{R}$.
Let’s just assume $f \in C^0([a, b])$ (this is closed and bounded)
Picture:



In the last class we went over the IVT (Intermediate Value Theorem) which gives us:
$f \in C^0([a, b])$, $f(a) < 0$, $f(b) > 0$, then $\exists x \in E$ such that $f(x) = 0$.

Def: signum e.g. `np.sign`,

- sgn(x) = 1 if $x > 0$
- sgn(x) = 0 if x = 0
- sgn(x) = -1 if $x < 0$

3.2.1 Bisection

```
import numpy as np
    a_0, b_0 = a, b
    k = 0
    while True:
        sign_left = np.sign(a_k)
        sign_mid = np.sign((a_k + b_k)/2)
        sign_right = np.sign(b_k)
        #check if any of the signs == 0
        if sign_left != sign_mid:
            a_{k+1}, b_{k+1} = a_k, (a_k + b_k)/2
            continue
        if sign_mid != sign_right:
            a_{k+1}, b_{k+1} = (a_k + b_k)/2, b_k
            continue
```

Listing 3.1: Bisection Pseudocode

3.2.2 Time Complexity (How fast is this?)

$\frac{|b_{k+1}-a_{k+1}|}{|b_k-a_k} = \frac{1}{2} \Rightarrow |b_k - a_k| = \frac{b-a}{2^k}$

Accuracy:
for rootfinding (solving $f(x) = 0$) there are 2 options:

1. $x - x_k$
2. $f(x) - f(x_k)$ ('residual')

Question: how many steps to get $|x - x_k| < \epsilon$?
$\Leftrightarrow$ how big does k need to be to get $|b_k - a_k| < \epsilon$

$$
\begin{aligned}
&= \epsilon = |b_k - a_k| = \frac{|b-a|}{2^k} \Rightarrow 2^k\epsilon = |b-a| \\
&\Longrightarrow log\epsilon + klog2 = log|b-a| \\
&= k = \frac{log\frac{1}{\epsilon} + log|b-a|}{log_2} = log_2\frac{1}{\epsilon} + log_2|b-a| \\
&= k = 0(log\frac{1}{\epsilon}) \\
&= \frac{log10^{10}}{log2} \approx 33
\end{aligned}
$$

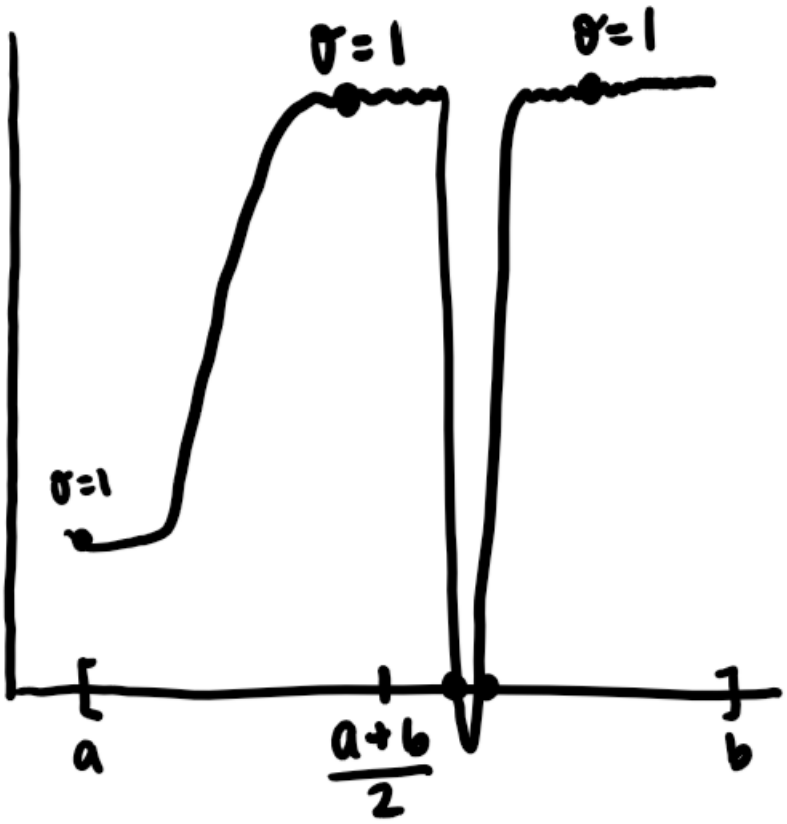
What should we assume about f?
1) Completely problem dependent
2) Easy Rootfinding problem

- Exercise: 1) how to solve p(x) = 0, if $p(x) = ax^2 + bx + c$
- Answer: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Let's say f(x) = $p(x) + \epsilon$(x)
What can we say about solving f(x) = 0?

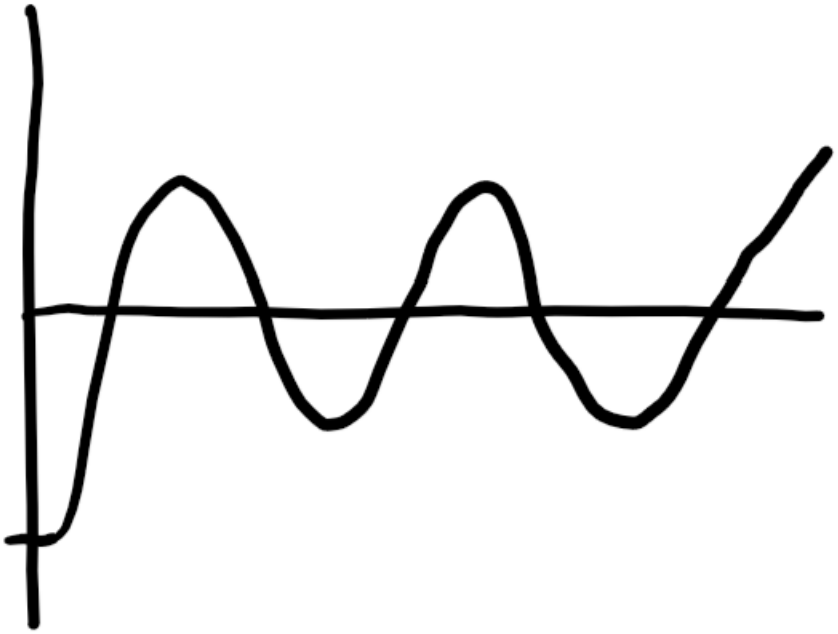
Professor's point: if $\epsilon(x)$ is small, then there is no need for the bisection algorithm, instead, we just use the quadratic formula.

3.2.3 Hard rootfinding problem



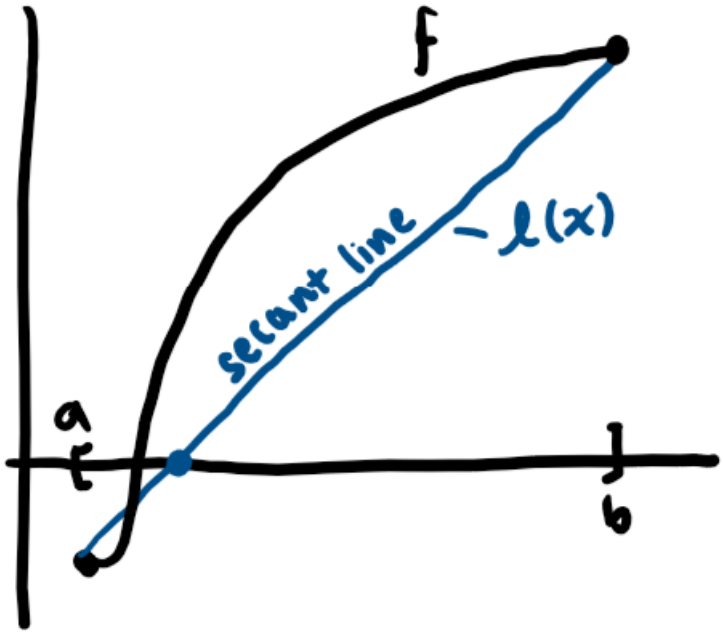
Also look at Weierstrass Monster function and the solution of a stochastic differential equation ('stock price graph')

e.g. This is a polynomial graph:



Then: Sturm's theorem gives us global information to guide our search . . .

How do we make bisection go faster? assuming that we have a reasonable problem to solve.



Exercise: what is $l(x)$?
Secant method:

```
x_0 = a
x_1 = b
k = 2
while True:
    x_k = x_{k-1} - f_k(x_k - x_{k-1}/(f_k - f_{k-1}))
    # check if |x_{k+1} - x_k| < tol
```

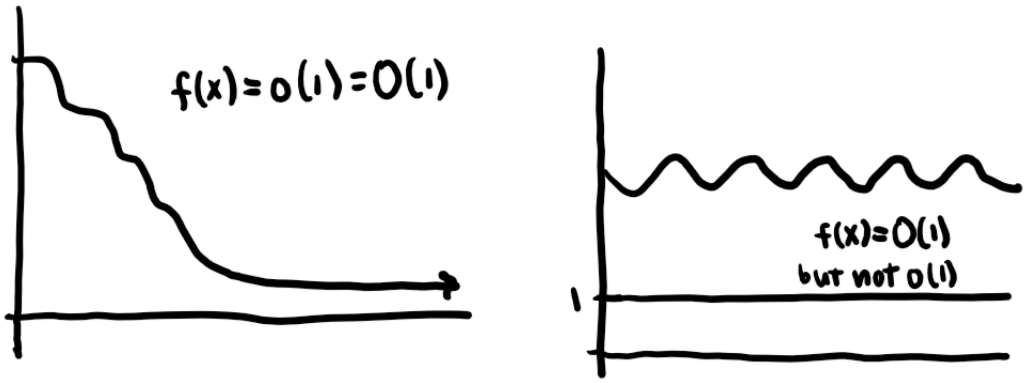
Listing 3.2: Secant method Pseudocode

Question: time complexity?

- rate of convergence
- orders of convergence

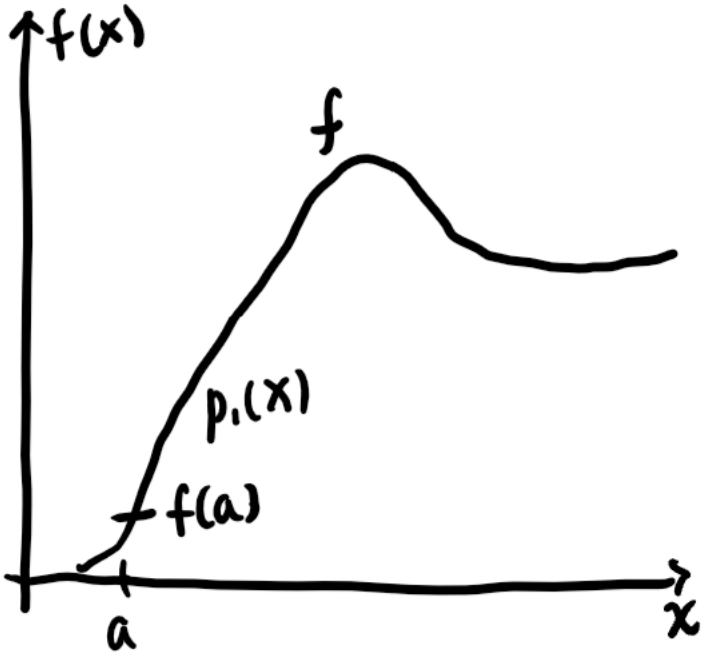
Landau notation: big O notation . . .
Def: $f(x) = O(g(x))[\text{as } x \to x]$
if: $lim_{x \to x_0} \frac{|f(x)|}{|g(x)|} = 0$

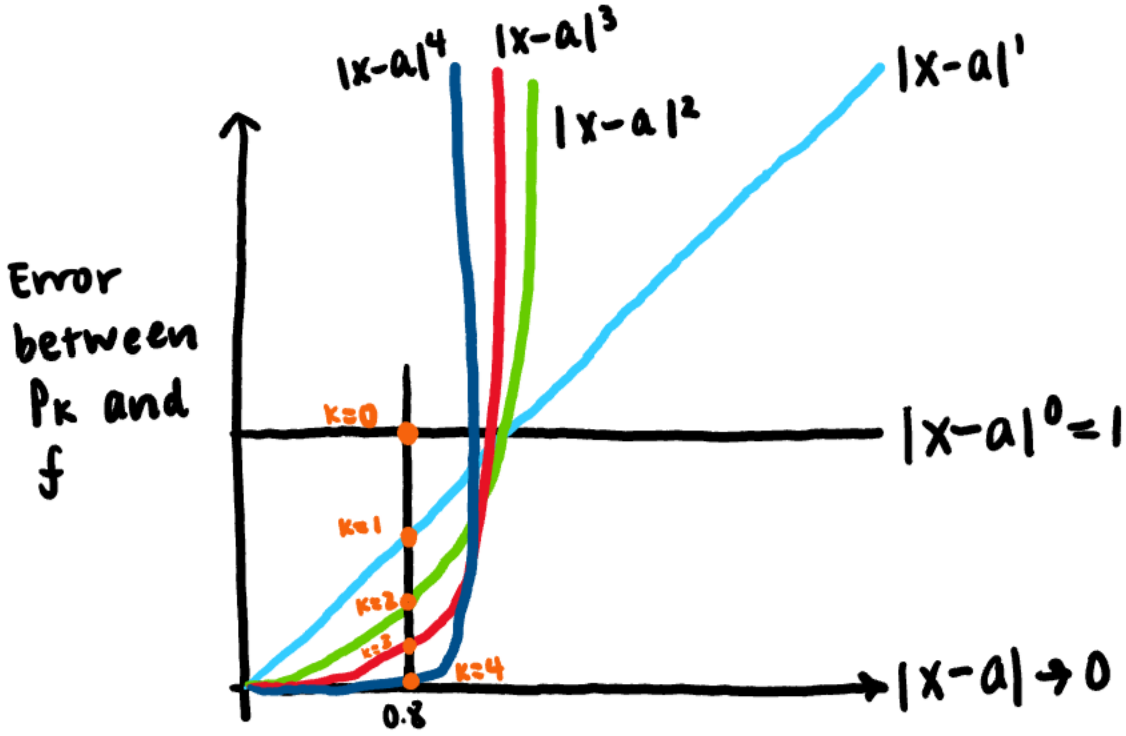
Some examples:



Def: let f: $\mathbb{R} \to \mathbb{R}$ be a C^∞ (infinitely differentiable function) in a ball surrounding a point $a \in \mathbb{R}$. Then there exists a polynomial (called the Taylor polynomial of order k).
$p_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f^{(3)}}{3!}(x-a)^3 + ... + \frac{f^{(k)}}{k!}(x-a)^k$
$= \sum_{m=0}^{k} \frac{f^{(m)}}{m!}(x-a)^m$
Such that: the remainder

$$R_k(x) = f(x) - p_k(x) = O(|x-a|^k) \qquad \text{as } x \to a$$





More useful forms of Taylor Expansions (TEs) for our purposes:

$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + O(h^3)$ [or $o(h^2)$]

- $O(h^3)$ is for bookkeeping to keep track of leftover error

$O = f(x) = f(x - h + h)$

- x - h is the base point TE

linearizing f about $x - h = f(x-h) + f'(x-h)h + O(h^2)$
Relabel: $x_k = x - h$
$x_k + 1 - x_k = x - x + h - h$

- $\triangle x_k$ is used as 'the step'

O = f(x_k) + f'(x_k)($x_{k+1} - x_k + (x_{k+1} - x_k)^2$ + O|$x_{k+1} - x_k|^2$
Rearrange:
$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} + O|x_{k+1} - x_k|^2$

1. Started with the equation we are solving
2. Taylor expanded the equation in a sensible way
3. Converted the equation into an iteration
4. Therefore we Taylor Expanded, we have an estimate of the error (more or less).

Exercise: In the derivation of Newton's method what happens if $f'(x_k) \sim x_{k+1} - x_k$

Chapter 4

Convergence of iterative methods in 1D

[SFP: Scribed by Mei Shin Lee.]

For solving equations $f(x) = 0$ we iterated using

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

Let's say we want to solve

$$x^* = \arg\min_{a \le x \le b} f(x)$$

If f is $C'([a,b])$ (continuous, differentiable on (a,b)) and its derivative f' is continuous, then a "first order necessary condition for optimality" is $f'(x^*) = 0$
In detail,

- "first order": Look at f'
- "necessary condition": Must be the interior point minimum. While this statement may be true, it doesn't always imply that it is a minimum.
- "optimality": minimum

A second order sufficient condition for minimization would be $f''(x^*) > 0$.
Lastly, to determine the behavior of the x^* at the endpoints, we will use Lagrange Multipliers.

What if we apply Newton's Method to the 1st order necessary condition?
Set $g(x) = f'(x)$.

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

What happens if $f(x) = ax^2 + bx + c$?

$$\begin{aligned} x_0 &= x... \text{ (doesn't matter)} \\ f'(x) &= 2ax + b \\ f''(x) &= 2a \\ x_1 &= x_0 - \frac{2ax + b}{2a} = \frac{-b}{2a} \end{aligned}$$

Now if $\frac{-b}{2a}$ is substituted into $f'(x)$, observe that

$$f'(\frac{-b}{2a}) = 2a(\frac{-b}{2a}) + b = 0$$

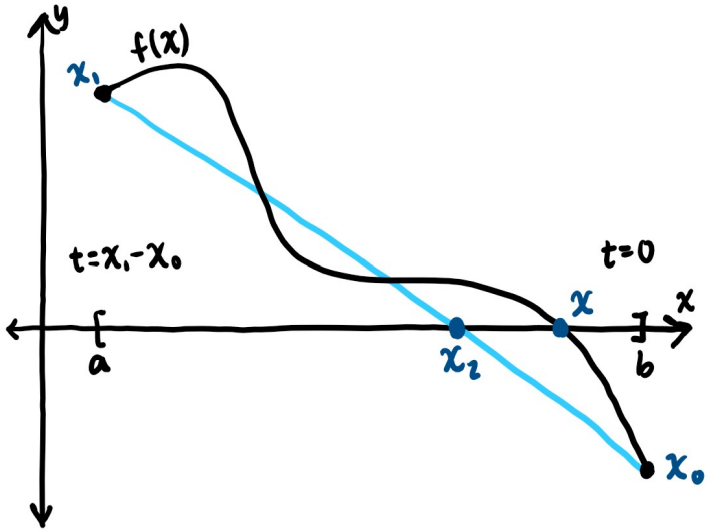
It converges in one step.

Exercise: Show that minimizing a function $f : \mathbb{R} \to \mathbb{R}$ using Newton's method is equivalent to minimizing a sequence of quadratics.
Hint: Do a $k = 2$ Taylor Expansion of $f(x_n + \Delta x_n)$ about x_n where $\Delta x_n = x_{n+1} - x_n$

Bonus Exercise: Apply the above exercise to the Secant Method.

4.1 Convergence of the Secant Method

Recall the Secant Method, where we solve for $f(x) = 0$



$$\begin{aligned} l(t) &= \frac{f_1 - f_0}{x_1 - x_0}(t) + f_0 \\ l(0) &= f_0 \\ l(x_1 - x_0) &= \frac{f_1 - f_0}{x_1 - x_0} + f_0 \\ &= f_1 - f_0 + f_0 \\ &= f_1 \end{aligned}$$

Now, find t such that $l(t) = 0$

$$t = -f_0 \cdot \frac{x_1 - x_0}{f_1 - f_0}$$
$$x_2 - x_1 = -f_0 \cdot \frac{x_1 - x_0}{f_1 - f_0}$$
$$x_2 = x_1 - f_0 \cdot \frac{x_1 - x_0}{f_1 - f_0}$$
$$x_{n+1} = x_n - f_n \cdot \frac{x_n - x_{n-1}}{f_n - f_{n-1}}$$

The derived equation: $x_{n+1} = x_n - f_n \cdot \frac{x_n - x_{n-1}}{f_n - f_{n-1}}$ is the secant iteration.

How do we address the question of how fast a sequence approach its limit?

Definition: Let $\xi = \lim_{x\to\infty} x_n$. Then we say that $x_n \to \xi$ with order $q > 1$ if the sequence $\varepsilon_n = |\xi - x_n|$ (difference of current iterate and target) converges to 0 and there exists some $\mu \in (0, 1)$ such that

$$\lim_{x\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^q} = \mu$$

If $q = 1$, we say that it converges linearly. If $q = 2$, it converges quadratically. Newton's method converges with order 2.

Theorem: If the secant method converges, then it converges with the rate $q = \frac{1+\sqrt{5}}{2}$.
Proof: Assume $x_n \to \xi$ and $\varepsilon_n = x_n - \xi$
We have the secant iteration

$$x_{n+1} = x_n - f_n \cdot \frac{x_n - x_{n-1}}{f_n - f_{n-1}}$$

Subtract ξ from both sides:

$$\begin{aligned}
\varepsilon_{n+1} &= \varepsilon_n - f_n \cdot \frac{x_n - x_{n-1}}{f_n - f_{n-1}} \\
&= \varepsilon_n - f_n \cdot \frac{\varepsilon_n - \varepsilon_{n-1}}{f_n - f_{n-1}} \\
&= \frac{\varepsilon_n(f_n - f_{n-1}) - f_n(\varepsilon_n - \varepsilon_{n-1})}{f_n - f_{n-1}} \\
&= \frac{f_n\varepsilon_{n-1} - \varepsilon_n f_{n-1}}{f_n - f_{n-1}}
\end{aligned}$$

Remember: $x_n = \varepsilon_n + \xi$ and f_n is just notation for $f(x_n)$. So we Taylor expand about ξ.

$$\begin{aligned}
f_n &= f(x_n) = f(\varepsilon + \xi) \\
&= f(\xi) + \varepsilon_n f'(\xi) + \varepsilon_{\frac{n}{2}}^2 f''(\xi) + O(\varepsilon_n^3) \\
&= 0 + \varepsilon_n f'(\xi) + \varepsilon_{\frac{n}{2}}^2 f''(\xi) + O(\varepsilon_n^3)
\end{aligned}$$

Write $f^{(p)}(\xi) = f_*^{(p)}$. (So, $f(\xi) = f_*$, $f'(\xi) = f'_*$..., etc.)
Taylor expand f_n (done above) and f_{n-1} about ξ to get:

$$f_n = \varepsilon_n f'_* + \frac{\varepsilon_n^2}{2} f''_* + O(\varepsilon_n^3)$$

$$f_{n-1} = \varepsilon_{n-1} f'_* + \frac{\varepsilon_{n-1}^2}{2} f''_* + O(\varepsilon_{n-1}^3)$$

Now substitute: $f'_* = f'(\xi)$

$$\begin{aligned}
\varepsilon_{n+1} &= \frac{[(\varepsilon_n f'_* + \frac{\varepsilon_n^2}{2} f''_* + O(\varepsilon_n^3))\varepsilon_{n-1} - (\varepsilon_{n-1} f'_* + \frac{\varepsilon_{n-1}^2}{2} f''_* + O(\varepsilon_{n-1}^3))\varepsilon_n]}{\varepsilon_n f'_* + \frac{\varepsilon_n^2}{2} f''_* + O(\varepsilon_n^3) - \varepsilon_{n-1} f''_* - \frac{\varepsilon_{n-1}^2}{2} f''_* + O(\varepsilon_{n-1}^3)} \\
&= \frac{\varepsilon_n \varepsilon_{n-1} [\frac{\varepsilon_n - \varepsilon_{n-1}}{2} f''_* + O(\varepsilon_n^2) + O(\varepsilon_{n-1}^2)]}{(\varepsilon_n - \varepsilon_{n-1}) f'_* + O(\varepsilon_n^2) + O(\varepsilon_{n-1}^2)} \\
\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n \varepsilon_{n-1}} &= \lim_{n\to\infty} \frac{[\frac{\varepsilon_n \varepsilon_{n-1}}{2} f''_* + O(\varepsilon_n^2)]}{[(\varepsilon_n \varepsilon_{n-1}) f'_* + O(\varepsilon_n^2)]} \\
&= \frac{f''_*}{2f'_*}
\end{aligned}$$

We need to find a constant $\mu > 0$ such that $\frac{\varepsilon_{n+1}}{\varepsilon_n^2} \to \mu$ as $n \to \infty$
Let's just say that $\varepsilon_{n+1} = C\varepsilon_n^q$

$$\begin{aligned}
\frac{\varepsilon_{n+1}}{\varepsilon_n \varepsilon_{n-1}} &= \frac{C\varepsilon_n^q}{\varepsilon_n \varepsilon_{n-1}} \\
&= \frac{C[C\varepsilon_{n-1}^q]^q}{C\varepsilon_{n-1}^q \varepsilon_{n-1}} \\
&= \frac{C \cdot C^q \cdot q_{n-1}^{q^2}}{C\varepsilon_{n-1}^{q+1}} \\
&= C^q \varepsilon_{n-1}^{q^2-q-1} \\
\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n \varepsilon_{n-1}} &= \lim_{n\to\infty} C^q \varepsilon_{n-1}^{q^2-q-1}
\end{aligned}$$

Now we pick q such that $q^2 - q - 1 = 0$

$$q = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2} < 0$$

Choose

$$q = \frac{1 + \sqrt{5}}{2}$$

This argument shows that if the Secant Method converges, it does so with order $q = \frac{1+\sqrt{5}}{2}$.

4.2 A Common Pattern with Taylor Expansions

1. Do two similar Taylor Expansions
2. Look for cancellations with every other power

Example:

$$F(x+h) = F(x) + hF'(x) + \frac{h^2}{2}F''(x) + O(h^3)$$
$$F(x-h) = F(x) - hF'(x) + \frac{h^2}{2}F''(x) + O(h^3)$$
$$F(x+h) - F(x-h) = 2hF'(x) + O(h^3)$$

Rearrange:

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

The result looks similar to $F'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$. This is called a *finite difference approximation*.

4.3 Taylor Expansion Remainders

Another form is the so-called Lagrange form of the remainder. It looks like:

$$f(x+h) = \sum_{m=0}^{k} \frac{f^{(m)}}{m!}h^m + \frac{f^{(m+1)}\eta}{(m+1)!}h^{m+1}, \eta \in [0,h]$$

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + ... + \frac{f^{(m)}x_0}{m!}(x-x_0)^m + \frac{f^{(m+1)}\eta}{(m+1)!}(x-x_0)^{m+1}, \eta \in [x, x_0]$$

$$\begin{aligned} 0 = f(\xi) &= f(x_n + \xi - x_n) \\ &= f(x_n) + f'(x_n)(\xi - x_n) + \frac{f''(\eta)}{2}(\xi - x_n)^2 \end{aligned}$$

Rearrange and apply the Newton step to get:

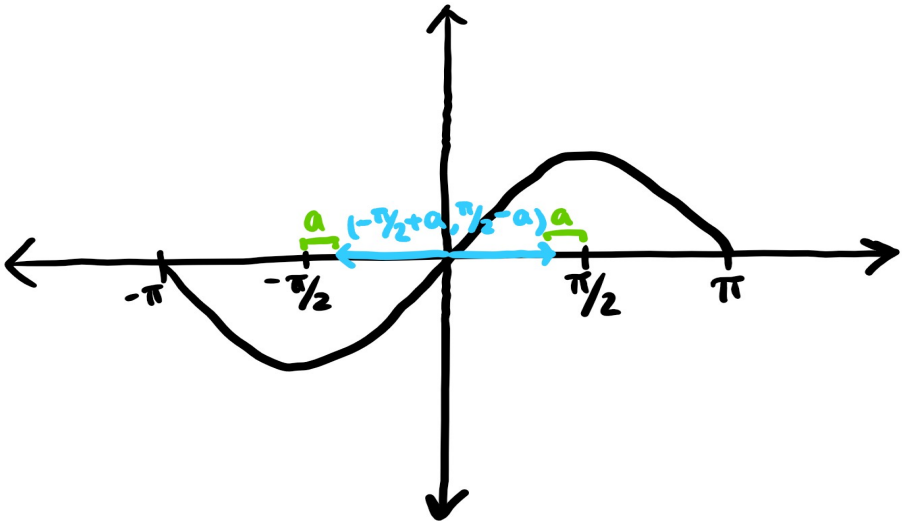
$$\xi - x_{n+1} = -\frac{(\xi - x_n)^2}{2} \cdot \frac{f''(\eta)}{f'(x_n)}$$
$$\lim_{n\to\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^2} = \frac{-1}{2} \cdot \frac{f''(\xi)}{f'(\xi)}$$
$$\to q = 2 \text{ for Newton}$$

4.4 Theorem for the Homework Problem (Q2)

Note: This ended up not being given for Written Homework #1 as per Professor Potter's email.

Theorem: Let $f \in C^2$ on $I_\delta = [\xi - \delta, \xi + \delta], \delta > 0$. Assume that $f(\xi) = 0$ and $f''(\xi) \neq 0$. Assume that $\exists A > 0$ such that $\frac{|f''(x)|}{|f'(y)|} \leq A$ for all $x, y \in I_\delta$. If x_0 is such that $|\xi - x_0| \leq min(\delta, \frac{1}{A})$, then $x_n \to \xi$ quadratically.

Exercise (HW): Let $f(x) = sin(x)$, so $sin(0) = 0$.



Apply this theorem to show that if $x_0 \in (\frac{-\pi}{2} + a, \frac{\pi}{2} - a)$, where $a \geq 0$ and $x_0 \neq 0$, $x_n \to 0$ quadratically.
Note: Do you have to assume anything about a?

4.5 Written Homework 1 Hint

The problem we want to solve is the convergence of $f(y) = \frac{1}{2}(\frac{x}{y} + y)$.
Our goal is to:

1. Fix $y_0 > 0$
2. If $y_0 < \sqrt{x}$, check the region where y_1 falls in, and choose $[a, b]$ wisely so that the Contract Mapping Theorem can be applied

Chapter 5

Different sources of error

[SFP: Scribed by Nigel Shen.]

5.1 Written Homework 1 Hint

The problem we want to solve is the convergence of $f(y) = \frac{1}{2}(\frac{x}{y} + y)$.
Our goal is to:

1. Fix $y_0 > 0$
2. If $y_0 < \sqrt{x}$, check the region where y_1 falls in, and choose $[a, b]$ wisely so that the Contract Mapping Theorem can be applied.

5.2 Four Major sources of error

1. Truncation error
2. Termination error
3. Statistical error
4. Round off error (Not covered in this lecture)

5.3 Truncation Error

Let's compute a finite difference approximation to the derivative:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + O(h^3)$$

Subtract $f(x)$ from both sides, divide by h, and rearrange, we get:

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

If we truncate and approximate $f'(x)$ with the $k = 1$ Taylor polynomial, we get:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

with truncation error:

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

This particular finite difference approximation is called a *forward difference*.

Exercise: With $x = \frac{\pi}{2}$, approximate $\frac{d}{dx}sin(x)$ at $x + h$ where $h = 0.1, 0.01, 0.001, ...$ What trend do you observe? Can you relate it to the values of $\frac{f''(x)}{2}$?

Def: The error in an approximation like this can be written as $Ch^p + O(h^p + 1)$, then the approximation is p^{th} order accurate.

Thus, we find that $f(x) \approx \frac{f(x+h)-f(x)}{h}$ is first-order accurate. If we did a *least square fit* of the errors, we would expect them to match Ch^1.

Now, in order to increase the accuracy, let's take the Taylor expansion with one more term:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f^{(3)}(x)\frac{h^3}{6} + O(h^4)$$

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f^{(3)}(x)\frac{h^3}{6} + O(h^4)$$

Subtract them, we get:

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{f^{(3)}(x)}{3}h^3 + O(h^4)$$

Rearrange terms, we get:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(x)}{6}h^2 + O(h^3)$$

This is called a *central difference*. It is second-order accurate.

Exercise:
1. What happens if $f^{(3)}(x) = 0$?
2. $p(x) = ax^2 + bx + c$. Show that the central difference scheme is "exact": That there is no error.

5.4 Termination Error

We saw two different kinds of iterative schemes in previous lectures:

* $x_{n+1} = f(x_n)$, which solves $f(x) = x$ to find the fixed points;
* $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ which solves $f(x) = 0$ (Newton's equation solving root-finding)

Exercise: The Secant method makes a linear approximation from x_n,x_{n-1},$f(x_n)$,$f(x_{n-1})$ at each step and finds that line's intersection with the x-axis to compute x_{n+1}. Newton's method uses f_n and f'_n. Write down a method similar to the secant method which makes a quadratic approximation using x_n, x_{n-1}, x_{n-2} and $f_n, f_{n-1} f_{n-2}$ and finds its intersection with the x-axis at each step.

Let's denote $e_n = x_n - x$ where x is the solution. This is the absolute error at each step. The method converges with order q accuracy if

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^q} = \mu \in (0, 1)$$

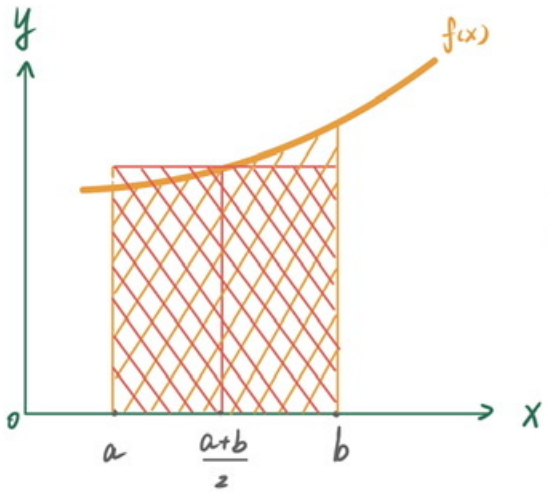
Note: e_n is as hard to compute as x_n. We don't know x. If we know e_n and x_n well, then we can find $x = x_n + e_n$ and we are done. We can instead look at $x_{n+1} - x_n$, which tells us if x_n converges.

5.5 Statistical Error

We'll see later in the class methods for approximating integrals ("numerical quadrature"). The idea comes from the derivation of Riemann integral:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

Mid-point Estimation: If the interval $[a, b]$ is not big, we might want to use Mid-point estimation: $\int_a^b f(x)dx \approx f(\frac{a+b}{2})(b - a)$



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

Exercise: Derive an error estimate for the midpoint rule applied to $\int_a^b f(x)dx$ assuming $b - a = O(h), h > 0$ and using a Taylor expansion.

For integrating multidimensional functions we can use tensor product quadrature:

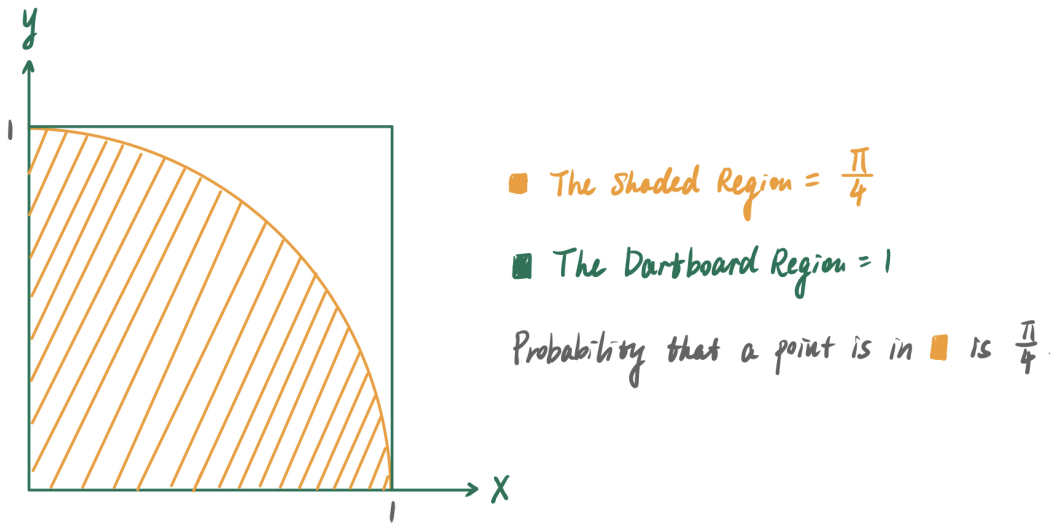
$$\int_{a_1}^{b_1}\int_{a_0}^{b_0} f(x,y)dxdy \approx \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} f(x_i, y_j)$$

Let's say we want to integrate a d-dimensional function using a $1D$ n-points quadrature rule, then the time complexity is $O(n^d)$. Therefore, instead people use *Monte-Carlo* methods.

Monte-Carlo Method: The idea is to generate a sequence of random approximations x_n approximating x. The error will frequency decay like $O(\frac{1}{\sqrt{n}})$ independent of dimension.

Example: Approximating π

The idea is to make a dartboard:



Let (x_n, y_n) be the position of the n^{th} dart throw, where $x_n, y_n \sim Uniform([0,1])$ and x_n, y_n are independent variables, so that our throws are independently and identically distributed.
After each throw we check whether it is in the shaded region:

$$h_n = \begin{cases} 1 & x_n^2+y_n^2 \leq 1 \\ 0 & otherwise \end{cases}$$

We expect that

$$\frac{\sum_{i=1}^{n} h_i}{n} \to \frac{\pi}{4}$$

Let $p = \frac{\pi}{4}$ and $p_n = \frac{h_n}{n}$. We then have

$$E[p_n - p] = E[p_n] - E[p]$$

$$= E[p_n] - p = E\left[\frac{\sum_{i=1}^{n} h_i}{n}\right] - p$$

$$= \frac{\sum_{i=1}^{n} E[h_i]}{n} - p = E[h_1] - p$$

$$= p - p = 0$$

Thus, p_n is an unbiased estimate.

Note that h_i is a Bernoulli random variable. That is, a random variable which has a probability p of being equal to 1, and a probability of $1-p$ of being equal to 0. The variance of a Bernoulli random variable is $p(1-p)$. In this case, $p = \pi/4$, so that:

$$Var(h_i) = \frac{\pi}{4}\left(1 - \frac{\pi}{4}\right), \qquad i = 1, \ldots, n. \tag{5.1}$$

Then we can compute:

$$Var(p_n - p) = \frac{1}{n^2}\sum_{i=1}^{n} Var(h_i) = \frac{1}{n^2}\cdot n \cdot \frac{\pi}{4}\left(1 - \frac{\pi}{4}\right) = \frac{\frac{\pi}{4}\left(1 - \frac{\pi}{4}\right)}{n}. \tag{5.2}$$

An alternative way of deriving this is by noticing that h_n is a Binomial random variable, whose variance is equal to $np(1-p)$, again where $p = \pi/4$.

Note that the standard deviation is in the original physical units of the problem, while the variance is not, which means that the standard deviation is used to measure the error of a Monte Carlo method. We have:

$$Std(p_n - p) = O(n^{-1/2}). \tag{5.3}$$

This "square root of $1/n$" rate of convergence is characteristic of Monte Carlo 3methods.

Def: Let Y be a quantity, and let $\hat{Y}$ be an approximation of Y. Then, the absolute error is

$$e = \hat{Y} - Y$$

and the relative error is

$$\varepsilon = \frac{\hat{Y} - Y}{Y}$$

Note:

1 The sign is correct in the sense that e.g. if $\hat{Y} = 4$ and $Y = 3$ then $e = 1$ means that $\hat{Y}$ is an overestimate

2 $\hat{Y} = Y + e = Y + Y\varepsilon = Y(1+\varepsilon)$, so $100 \cdot \varepsilon$ is the percentage error.

Chapter 6

Computer representation of numbers

[SFP: Scribed by Cindy Zhang, Feb 9.]

In this class, we are going to learn about how numbers are represented in computers. First, how do we represent a number? Let's answer the question in decimal, the base we are the most familiar with.

To represent a decimal number x, we can use this algebraic expression:

$$x = d_n \cdot 10^n + d_{n-1} \cdot 10^{n-1} + ... + d_0 \cdot 10^0 + d_{-1} \cdot 10^{-1} + d_{-2} \cdot 10^{-2} + ... \tag{6.1}$$

$d_i \in \{0, 1, ..., 9\}$, for all i.

Eg. $77.1 = 7 \cdot 10^1 + 7 \cdot 10^0 + 1 \cdot 10^{-1}$
Eg. $60.12 = 6 \cdot 10^1 + 0 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2}$

For additions, if we have two decimal numbers x and y, where:

$$x = x_n \cdot 10^n + ... + x_m \cdot 10^m, n \geq m$$

$$y = y_n \cdot 10^p + ... + y_q \cdot 10^q, p \geq q$$

Then:

$$x + y = z = z_r * 10^r + ... + z_s * 10^s \tag{6.2}$$

$$r \geq max(n, p)$$

$$s \geq min(m, q)$$

$$z_i = x_i + y_i$$

Problem: $x_i = 9, y_i = 9$ then $z_i = 18 = 1 \cdot 10^1 + 8 \cdot 10^0$.
Likewise: $xy = (x_n \cdot 10^n + ... + x_m \cdot 10^m)(y_p \cdot 10^p + ... + y_q \cdot 10^q)$
The idea is that we are representing x and y as rational function over 10. So instead of x^2+x+1, we have $10^2 + 10 + 1$.

To explicitly calculate product:

$$\begin{aligned}
77.1 \cdot 60.12 &= (7 \cdot 10^1 + 7 \cdot 10^0 + 1 \cdot 10^{-1})(6 \cdot 10^1 + 0 \cdot 10^0 + 1 \cdot 10^{-1} + 2 \cdot 10^{-2}) \\
&= 42 \cdot 10^2 + 0 \cdot 10^1 + 7 \cdot 10^0 + 14 \cdot 10^{-1} \\
&+ 42 \cdot 10^1 + 0 \cdot 10^0 + 7 \cdot 10^{-1} + 14 \cdot 10^{-2} \\
&+ 6 \cdot 10^0 + 0 \cdot 10^{-1} + 1 \cdot 10^{-2} + 2 \cdot 10^{-3} \\
&= 42 \cdot 10^2 + 42 \cdot 10^1 + 13 \cdot 10^0 + 21 \cdot 10^{-1} + 15 \cdot 10^{-2} + 2 \cdot 10^{-3} \\
&= 4 \cdot 10^3 + (2+4) \cdot 10^2 + (2+1) \cdot 10^1 + (3+2) \cdot 10^0 + (1+1) \cdot 10^{-1} + 5 \cdot 10^{-2} + 2 \cdot 10^{-3} \\
&= 4 \cdot 10^3 + 6 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2} + 2 \cdot 10^{-3} \\
&= 4635.252
\end{aligned}$$

We use base-10 arithmetic. Let's consider other common bases.
Other bases:

- Babylonians used base 60.
  Note: Using base 10 system provides us convenient intuitions about multiples of 2 and 5, which are factors of 10. Imagine for Babylonians, it's as easy to count multiples of 2 3 4 5.. as it is for us to count in multiples of 2 and 5.
- Computers use binary.
  Base 2. d = 0, 1. Why? Side thought: how to encode things in bits (of any base...)
  Potential reasons:
  - signal (electrical voltage)
  - Boolean: true/false
  - hardware
  - there are only two signs

The **radix economy** of a number N in any base b is:

$$b \cdot \lfloor log_b N + 1 \rfloor \tag{6.3}$$

$$\sim b \cdot log_b N \text{ for large N}$$

- $\sim$ = is asymptotic to

If we looked at the "relative radix economy"
= the radix economy divided by $log_e N$

$$\sim b \cdot \frac{log N}{log b} / log N = \frac{b}{log b} \tag{6.4}$$

b	2	3	4
$\frac{b}{logb}$	2.88...	2.73...	2.88...

Some other thoughts regarding ternary:

- fiber optic computer: no light, and two orthogonally polarized modes
- true/false/maybe

Question: balanced or unbalanced?

- unbalanced: {0, 1, 2}
- balanced: {-1, 0, 1}

6.1 Systematic Way of Encoding Rational Numbers

How to encode numbers on a computer?

Let's say we have a fixed-length word of length N of memory, in binary. Can we encode any real number in N bits?

Can't encode π because it's irrational. Has an infinite number of non-repeating digits.

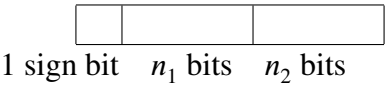
But ALL rational numbers with an infinite number of digits but which nevertheless can be encoded with a fixed number of bits.

For a fixed N, you can't encode ALL rationals, but for each rational, you can encode it using a finite number of bits by encoding its numerator and denominator.

Eg: rational approximation of π: $\frac{22}{7}$, $\frac{355}{113}$, $\frac{104348}{33215}$,...

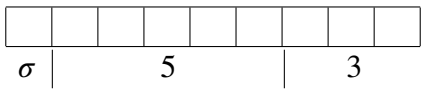
Idea 1: Fixed Point Representation

Allocate a fixed number of bits to encode the part of a number before the decimal place, and ditto for the part after:

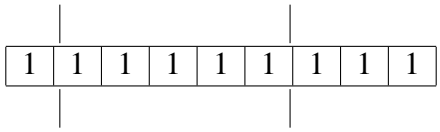


$\boxed{\text{n}_1 + n_2 + 1 \text{ total}}$

Eg:



$$x = \pm(b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + b_{-2} 2^{-2} + b_{-3} 2^{-3})$$



range: [-31.875, 31.875] ← numbers are equally spaced on this interval.

Eg: WAV format
one channel of audio in WAV format w/ sampling rate 44.1 kHz of length T seconds will be:
$\sim 44,100 \times T$ $44,100 \times T$ numbers, ideally between -1 and 1 (each number is called a sample)
Fixed point isn't totally appropriate here because human perception of sound loudness is logarithmic. It is wasting space to evenly allocate memory here, because we can spread logarithmically to optimize the range of numbers we can represent.

Idea 2: Floating Point
We write:

$$\pm m \times 2^{\pm e} \tag{6.5}$$

- $\pm$ = sign
- m = mantissa of significance
- $\pm$ = sign e = exponent

Allocate a fixed number of bits for each of these things.
Most important formats are those provided by IEEE754.
If you use a "float" or "double" in C on your computer, you will get a 32-bit floating point umber and 64-bit, respectively.
Note: normalize the mantissa so that $m \in [1, 2)$ (looks like 1, 001010...)

Eg: 32-bit single-precision float:

sign	e	1	m

1 8 bits 1 22 bits

(1+8+1+22=32)

$\boxed{\underline{\quad e \quad}}$: the sign is stored implicitly
The range of numbers that can be represented by 32 bits is $-2^8, ..., 2^8 - 1$ (two's complement number representation).
So: if $\boxed{1}$ (normalized) is actually $\boxed{0}$, then the number is called "denormal"

IEEE754: standardize a floating point format with consistent rules:
1) representation (# of bits) (32-bit, 64-bit, 80-bit)

2) correct and consistent rounding
3) handling exceptions. e.g. $\frac{1}{0}$

The most important IEEE754 rule for this lecture is:
e.g. for addition:

$$round(a+b) = a \bigoplus b \tag{6.6}$$

- $+$ = real $+$
- $\bigoplus$ = computer $+$

$$= (a+b)(1+\delta)$$

- δ = relative error

$$|\delta| < |\epsilon_{mach}|$$

Def: machine epsilon , "ϵ_{mach}", for a floating point format is the difference between the next biggest number after 1 and 1.

Eg:
$1.0 = (1 \mid 0 \ldots 0 \mid 1 \mid 0 \ldots 0)$
$1.0 + \epsilon_{mach} = (1 \mid 0 \ldots 0 \mid 1 \mid 0 \ldots 0\ 1)$

In single-precision: $2^{-23} \approx 1.19 \cdot 10^{-7}$
Double-precision (64-bit format):

- sign 1 bit
- exponent: 11 bit
- matissa: 52 bits (normalized)

$$\epsilon_{mach}(double) = 2^{-52} 1.1 * 10^{-16} \tag{6.7}$$

Exercise: how to round $\frac{1}{10}$?
Represent $\frac{1}{10}$
Hint:
$\frac{1}{10} = \frac{1}{2+2^3} = \frac{1}{2^3} \cdot \frac{1}{1+2^{-2}}$
Geometric series $= \frac{1}{2^3} \cdot (1 - 2^{-2} + 2^{-4} - 2^{-6} + \ldots)$
Do the rounding both ways, and compare with what your computer does.

```
round(0.1) = 0.1000 000 000 000 000 1
```

Chapter 7

Linear Algebra Review

[SFP: Scribed by Xinyu Gao.]

7.1 Definition

Let us say we want to solve a linear system:

$$\begin{cases} a_{11}x_1 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + ... + a_{2n}x_n = b_2 \\ \qquad\qquad\qquad\qquad ... \\ a_{m1}x_1 + ... + a_{mn}x_n = b_m \end{cases} \tag{7.1}$$

In this linear system, we have m equations and n unknowns. We can rewrite the linear system in "matrix form":

$$\begin{bmatrix} a_{11} & ... & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & ... & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \tag{7.2}$$

where the first matrix is an $m \times n$ matrix, the second matrix is a $n \times 1$ column vector, and the third matrix is a $m \times 1$ column vector.

We could also use the notation, where $A \in R^{m\times n}$,$x \in R^{n\times 1}$,$b \in R^{m\times 1}$, $Ax = b$.

It's helpful yo think of matrices as elements in a vector space. For example, for scales $\alpha, \beta \in R$ and matrices $A, B \in R^{m\times n}$, we have: $(\alpha A + \beta B)_{ij} = \alpha A_{ij} + \beta B_{ij}$ All this extends what we are familiar or comfortable with from R^n to a new vector space. Note: There is a natural(obvious) identification between the vector space R^n, $R^{n\times 1}$, and $R^{1\times n}$:

$R^n \rightarrow$"vectors", where $\vec{x} = (x_1, ..., x_n)$ tuple.

$R^{n\times 1} \rightarrow$"column vectors",

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$R^{1\times n} \rightarrow$"row vectors", $\begin{bmatrix} x_1 ... x_n \end{bmatrix}$

7.2 Matrix Multiplication

Matrix Multiplication is an operation which takes matrices from possibly matrix vector space and maps them to a third provided that they have compatible shapes.

So, specially, if $A \in R^{m\times p}$ and $B \in R^{p\times n}$, then their product AB is in $R^{m\times n}$. Its components are given by the following rule:

$$(AB)_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj}$$

, where $(AB)_{ij}$ means the ith row and the jth column.

Let's evaluate AB in matrix form:

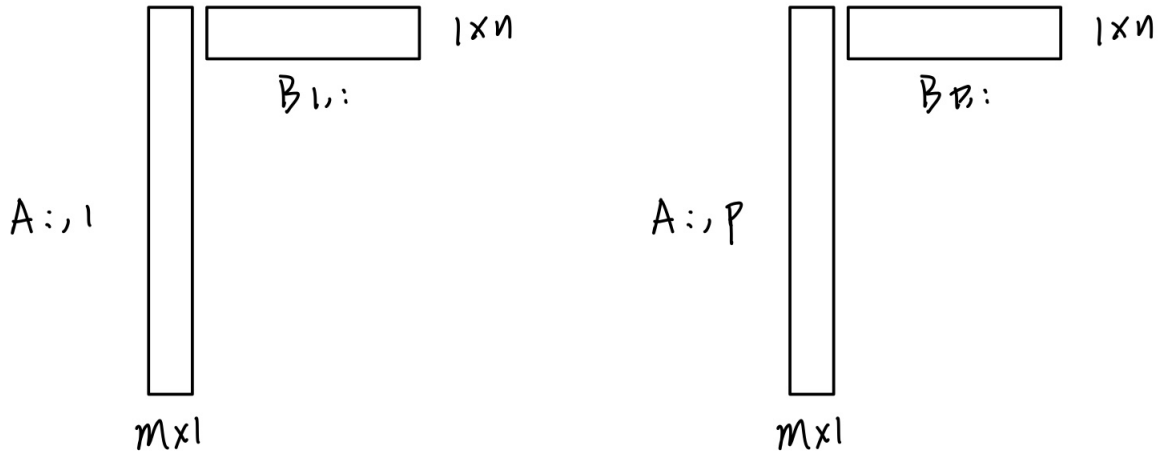
$$\begin{bmatrix} AB_{11} & \dots & AB_{1n} \\ \vdots & \vdots & \vdots \\ AB_{m1} & \dots & AB_{mn} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{p} A_{1k}B_{k1} & \dots & \sum_{k=1}^{p} A_{1k}B_{kn} \\ \vdots & \vdots & \vdots \\ \sum_{k=1}^{p} A_{mk}B_{k1} & \dots & \sum_{k=1}^{p} A_{mk}B_{kn} \end{bmatrix} = \begin{bmatrix} A_{1,:}B_{:,1} & \dots & A_{1,:}B_{:,1} \\ \vdots & \vdots & \vdots \\ A_{1,:}B_{:,1} & \dots & A_{1,:}B_{:,1} \end{bmatrix} \in R^{m\times n} \tag{7.3}$$

where $A_{1,:}$ is a Matlab notation, representing the first row of A.

Remember: for $\vec{a}, \vec{b} \in R^n$, $\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cos(\theta)$, where θ is the angle between $\vec{a}$ and $\vec{b}$

Exercise: show that AB = $A_{:,1}B_{1,:} + \dots + A_{:,p}B_{p,:}$.

Figure 1. $AB = A_{:,1}B_{1,:} + \cdots + A_{:,p}B_{p,:}$



Note that if $a \in R^{m\times 1}, b \in R^{1\times n}$, then $ab \in R^{m\times n}$

Exercise: Write in python-style pseudo-code, an algorithm for multiplying the matrices $a \in R^{m\times p}, b \in R^{p\times n}$. How many floating point adds and multiplies do you need, in terms of m,n and p?

7.3 Matrices and geometric objects

We can think of matrices as geometric objects. Let's say we have n vectors $\vec{a}_i \in R^m$. Let's stack them as column vectors into a matrix $A \in R^{m \times n}$.

A = $\left[\; x_1 ... x_n \; \right]$

Let's look at the matrix vector product

$$A\vec{\alpha} = \vec{x}, \vec{\alpha} \in R^n, \vec{X} \in R^m$$

How do we think of this?

$$\vec{x} = A\vec{\alpha} = [\vec{a}_1 ... \vec{a}_n] \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \alpha_1 \vec{a}_1 + ... + \alpha_n \vec{a}_n$$

, which is the linear combination of vectors in R^m.

7.4 Transpose of a matrix

The transpose of a matrix, A^T is the matrix:

$$A^T = \begin{bmatrix} a_{11} & ... & a_{m1} \\ \vdots & \vdots & \vdots \\ a_{1n} & ... & a_{mn} \end{bmatrix} \in R^{n \times m}$$

What we did:

1.Flipped the matrix over its diagonal(a_{ii})

2.Or, swapped the indices $A^T_{ij} = A_{ji}$

Consequence: If $A \in R^{m \times n}$, then $A^T \in R^{n \times m}$

Sidebar: Often in physical problems, if you have a matrix A, the thing you multiply on its right could very well have different physical units than the thing you multiply on its left.

Now, what is $A^T \vec{y} = \vec{\beta}$?

$$\vec{\beta} = A^T \vec{y} = \begin{bmatrix} a_1 \cdot \vec{y} \\ \vdots \\ a_n \cdot \vec{y} \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

What about $AA^T \vec{z}$?

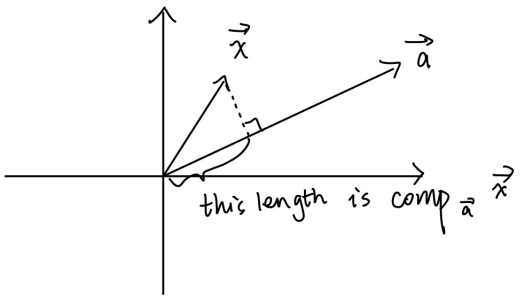
$$A^T \vec{z} = \vec{\gamma} = \begin{bmatrix} a_1 \cdot \vec{z} \\ \vdots \\ a_n \cdot \vec{z} \end{bmatrix}$$

Thus,

$$AA^T \vec{z} = (\vec{a}_1 \cdot \vec{z}_n + ... + \vec{a}_n \cdot \vec{z}_n)$$

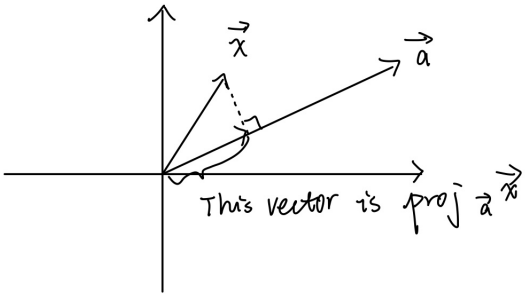
Recall, the scalar projection of $\vec{x}$ onto $\vec{a}$ is:$comp_{\vec{a}} \vec{x} = \frac{\vec{a}}{|\vec{a}|} \cdot \vec{x}$

Figure 2. Scalar projection of $\vec{x}$ onto $\vec{a}$



and, the vector projection $\vec{x}$ onto $\vec{a}$ is:$proj_{\vec{a}}\vec{x} = comp_{\vec{a}}\vec{x} \cdot \frac{\vec{a}}{|\vec{a}|} = (\frac{\vec{a}}{|\vec{a}|} \cdot \vec{x})\frac{\vec{a}}{|\vec{a}|}$

Figure 3.



Important idea: matrix agebra can easily encode geometric operations. You can take the matrix AA^T and think of it concretely in terms of a type of projection.

Exercises: Let $r(t) = r_0 + t\hat{d}$ where $r_0 \in R^3$ and $d \in R^3$ is a unit vector. Let $x \in R^3$ be an arbitrary point.

Find the closet point on the line: $\{r(t) : r \in R\}$ to the point x.

Hint: Solve $\min_t ||r(t) - x||^2$, where $||x|| = \sqrt{x_1^2 + ... + x_n^2}$

Then, how does this relate to projections?

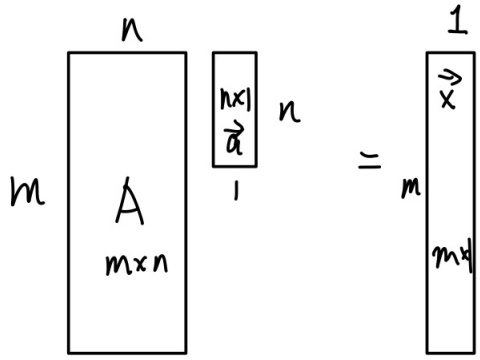
7.5 The linear system and three conditions

Let's say we have vector $\vec{a^1}, ..., \vec{a^n} \in R^m$ and another vector $\vec{x}$. How do we find coefficient $\vec{\alpha} = (\alpha_1, ..., \alpha_n)$ such that $\vec{x} = \alpha_1\vec{a}_1 + ... + \alpha_n\vec{a}_n$?

Solve the linear system: A lot depends on m and n:

Case1: n<m: "overdetermined case"

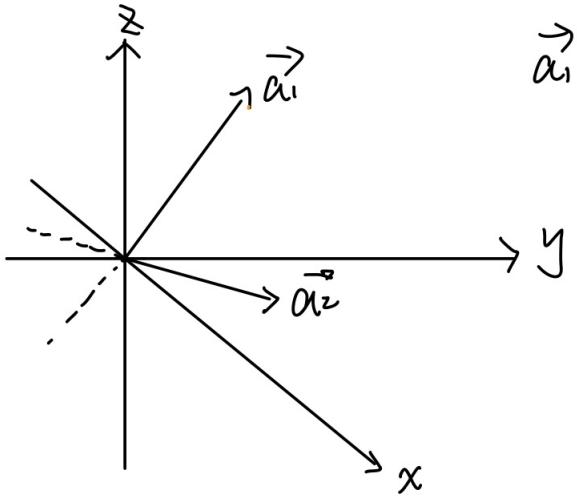
Figure 4.



Simple failure mode: $m = 3, n = 2, A = [\vec{a}_1 \vec{a}_2] \in R^{3\times 2}, \vec{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \in R^{2\times 1}, x \in R^3$.

This is all in 3-space(R^3),

Figure 5.



$\vec{a_1}$ & $\vec{a_2}$ generates an 2D plane

If we take linear combination of $\vec{a}_1$ and $\vec{a}_2$, $\alpha_1\vec{a}_1 + \alpha_2\vec{a}_2$ for all $\alpha_1, \alpha_2 \in R$, this generates a 2D plane in R^3, provided that there doesn't exist a constant $c \neq 0$, such that $\vec{a}_1 = c\vec{a}_2$.

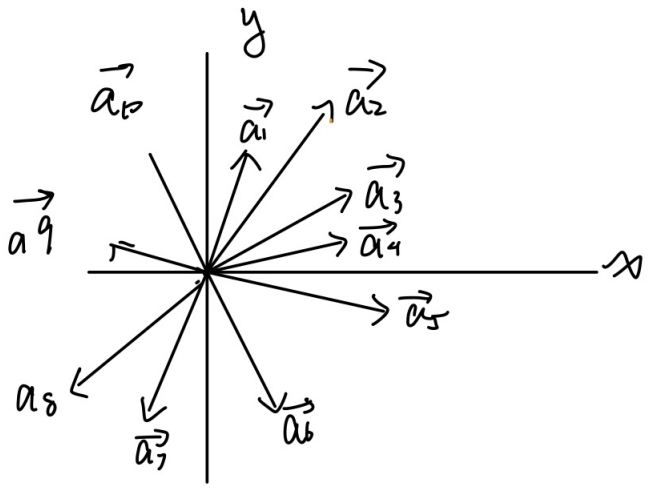
If $\vec{x}$ doesn't lie on that plane, then definitionally, there are no scales α_1, α_2 such that $\vec{x} = \alpha_1\vec{a_1} + \alpha_2\vec{a_2}$, hence we connot solve $A\vec{\alpha} = \vec{x}$ for $\vec{\alpha}$.

Case2: n>m :"Underdetermined"

This is a more subtle and interesting failure mode.

For example, $m = 2, n = 10$, the picture is:

Figure 6.



There are many solutions. How do we even begin to talk about which solution is best? An example is the sparse approximation.

Case3: n=m or A is a square metrix:"Underdetermined"

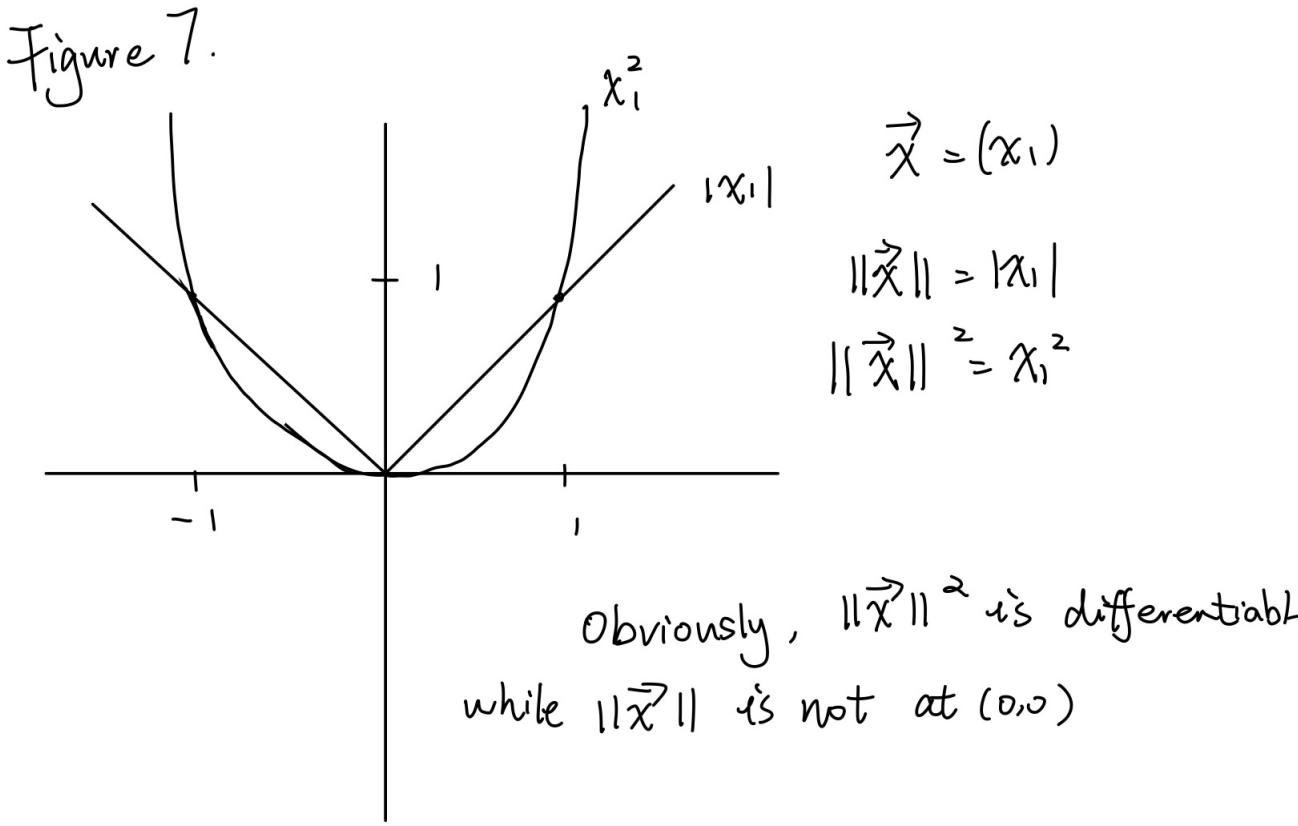
What we care about is the linear independence.

Definition of linear independence:$A\vec{\alpha} = \vec{x}$ where $A \in R^{N\times N}$ can be solved for $\vec{\alpha}$ if the column vectors of A are linearly independent. That means that there is no coefficient vector $(\alpha_1, ..., \alpha_n)$ with not all α_i such that $\alpha_1\vec{a_1} + ... + \alpha_n\vec{a_n} = \vec{0}$

Note: "invertible matrix theorem" is equivalent to $det(A) \neq 0, dim(null(A)) = 0, A^{-1}$ exists, such that $A^{-1}A = AA^{-1} = I$.

Exercise:Find (α_1, α_2) such that $||\vec{x} - \alpha_1\vec{a_1} - \alpha_2\vec{a_2}||^2$ is minimized.

Why minimize $||\vec{x}||^2$ instead of $||x||$? Picture in 1D.



We are able to see that $||\vec{x}||^2$ is differentiable while $||\vec{x}||$ is not.

7.6 Solve the linear system when n = m

How can we solve a system equations? We use Gaussia elimination.

It's easy to compute A^{-1}, for example np.lialg.inv(A), in a programming language. DO NOT USE THIS!

Let's say we want to solve:

$x_1 + x_2 + x_3 = 6$ $2x_1 + 4x_2 + 2x_3 = 16$ $-x_1 + 5x_2 - 4x_3 = -1$ How we do it: first we rewrite like this:

$$\left[\begin{array}{c|c} A & B \end{array}\right] = \left[\begin{array}{ccc|c} 1 & 1 & 1 & 6 \\ 2 & 4 & 2 & 16 \\ -1 & 5 & 4 & -3 \end{array}\right]$$

Use elementary row operations to trans from [A|B] into row echelon form, then solve to find (x_1, x_2, x_3).

There are three ways of "elementary row operations:"

1.add a row to another

2.multiply a row by constant

3.swap 2 rows
(1 and 2 can be done simultaneously)
There are three rules of "row echelon form":
1.All rows in the bottom should only contains zero.
2.The leading(first) non-zero number is one.
3.Every leading one is to the right of the one in each preceding row.

Chapter 8

LU Factorization

[SFP: Scribed by Chuanyang Jin.]

8.1 Introduction

The goal of today's lecture is to find an accurate algorithm for computing the LU factorization. Let's say we have a lower-triangular matrix

$$L = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix}. \tag{8.1}$$

How to solve $Lx = b$ for x?
We can first examine a simple example when $n = 2$:

$$\begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{8.2}$$

And rewrite this as a system of equations:

$$\begin{cases} l_{11}x_1 = b_1 \\ l_{21}x_1 + l_{22}x_2 = b_2 \end{cases} \tag{8.3}$$

We can solve it with the following two steps.

Step 1: Solve $x_1 = \frac{b_1}{l_{11}}$
Step 2: Solve $x_2 = \frac{b_2 - l_{21}x_1}{l_{22}}$

Note: The method above only works when $l_{11} \neq 0$ and $l_{22} \neq 0$. If $l_{11} = 0$ or $l_{22} = 0$, then the determinant of the matrix equals to 0, and the system cannot be solved.

8.2 Lower Triangular Solve

Can we extend this “algorithm" to a real algorithm for solving an $n \times n$ lower triangular matrix? The answer is: Yes.

Following the same idea, we can get:

$$x_i = \frac{b_i - \Sigma_{j=1}^{i-1} l_{ij} x_j}{l_{ii}} \tag{8.4}$$

Note: We interpret the sum $\Sigma_{j=1}^{i-1} l_{ij} x_j$ as 0 if $i = 1$.
The pseudo-code for the lower triangular solve is as follows:

Algorithm 1 Lower Triangular Solve

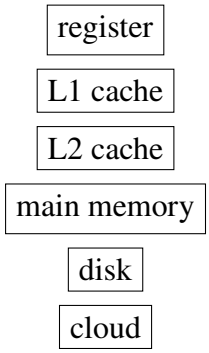
```
x_1 = b_1 / l_11
for i = 2, ..., n do
    x_i = b_i
    for j = 1, ..., i − 1 do
        x_i = x_i − l_ij x_ii
    end for
    x_i = x_i / l_ii
end for
```

Observations:

1. There is a “loop-carried dependency".
2. b plays kind of a minimal role.
3. Time: $O(n^2)$, extra space: $O(n)$.
4. There are potential accuracy problems.
   For example, approaching the matrix below may encounter an accuracy problem.
   $$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 10^{-7} \end{bmatrix}$$

Note:

Memory hierarchy



The speed increases from the bottom to the top.
The space increases from the top to the bottom.

8.3 In-place Lower Triangular Solve

An *"in-place" algorithm* is an algorithm that only uses $O(1)$ extra space to run. We can improve the Lower Triangular Solve by transforming it into an in-place version.

The key is to overwrite b instead. The pseudo-code is as follows:

Algorithm 2 Lower Triangular Solve (in-place version)

$b_1 = b_1/l_{11}$
for $i = 2, ..., n$ **do**
  for $j = 1, ..., i-1$ **do**
    $b_i = b_i - l_{ij}b_j$
  end for
  $b_i = b_i/l_{ii}$
end for

Note: Don't do this in Python. It would be extremely slow.
How about in numpy?

Algorithm 3 Lower Triangular Solve (numpy version)

for $i = 2, ..., n$ **do**
  $b_i = (b_i - l_{i,1:i-1} \cdot b_{1:i-1})/l_{i,i}$
end for

Note: Watch out for the difference between the 1-based indexing and 0-based indexing.
You have to be proficient at translating between them.
- 1 based: math, FORTRAN, MATLAB, Julia, etc.
- 0 based: math, Python, C, etc.

Note: This is a row-oriented lower triangular solve. Another thing to watch out for is whether a language stores matrices (or multi-dimensional arrays) in a row-major or column-major format.

Let's say we have a 2D array (a matrix perhaps)

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix}. \tag{8.5}$$

The computer memory for it is linear. If we arrange the array row-by-row, then it is called *"row-major"*. If we arrange it column-by-column, then it is called *"column-major"*.

- row major: numpy (default), C, etc.
- column major: numpy, FORTRAN, MATLAB, etc.

Try to think: what is numpy actually doing?

A = np.random.randn((m,n)) (randn: normally distributed random numbers)
B = np.random.rand((n,p)) (rand: uniformly distributed random number)
C = A @ B (@: BLAS, same as np.dot(A,B))

BLAS is the abbreviation of Basic Linear Algebra Subsystem.
There are multiple implementations of BLAS, such as OpenBLAS, GotoBLAS, IntelMKL, etc.

GotoBLAS was made by Kazushige Gotō (one of Prof. Potter's personal heroes).
Some anecdotes about him:
- worked at a patent office
- had long train commute
- wrote a hand-optimized implementation of BLAS on his train ride
- made contact with researchers doing numerical linear algebra
- for a while, his implementation was a lot faster than anything else

Exercise 1 We did a row-oriented "single for loop" (or "dot product style") in-place "forward solve" (i.e. lower triangular matrix solve). Please derive a row-oriented "single for loop" (or "dot product style") in-place "back solve" (i.e. upper triangular matrix solve).

Exercise 2 Count the FLOPs (floating-point operations).

Exercise 3 For your back solve, how can you simplify (or optimize) it if we apply it to a unit upper triangular matrix?

Exercise 4 If we want to efficiently solve $UX = B$, where $U \in R^{n\times p}$, $B \in R^{n\times p}$, how do we modify our algorithm? In other words, how do we do it in one back solve instead of p back solves? [Multiple RHS's]

Exercise 5 Prove that the product of two lower (resp. upper) unit triangular matrices is unit lower (resp. upper) triangular.

8.4 LU Factorization

Theorem. *Let $A \in R^{n\times n}$ satisfies that $det(A_{1:k,1:k}) \neq 0$ for $k = 1, ..., n-1$, then there exists a unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$.*

And if $det(A) \neq 0$, then L and U are unique.

Note: The concept of matrix factorization is very important.

Ideas: 1) Write A as a product of some factors. 2) But not just any factors. They should be highly structured and tell us something important about A (usually coming from a matrix subspace, group, manifold, etc.). 3) Ideally, the matrix factorization will lead to powerful algorithms.

For LU Factorization, we know that:

1) If A is non-singular, then $l_{ii} \neq 0$ and $u_{ii} \neq 0$ for all i. Then L and U come from groups of lower and upper triangular matrices.

2) Solving a linear system takes $O(n^3)$ time in general (e.g. use Cramer's rule). A method using LU factorization is as follows:

$$Ax = b$$
$$LUx = b$$
$$Ux = L^{-1}B$$
$$x = U^{-1}L^{-1}b$$

$L^{-1}b$ needs forward solve with runtime $O(n^2)$.
$U^{-1}L^{-1}b$ needs back solve with runtime $O(n^2)$.
But it takes $O(n^3)$ time to compute an LU decomposition in general.

Then how to compute LU?
We know three elementary row operations of matrices:

$$\text{“row add"} = \begin{bmatrix} 1 & ... & ... & ... \\ ... & 1 & ... & ... \\ \vdots & \alpha_{ij} & \vdots & \vdots \\ ... & ... & ... & 1 \end{bmatrix}$$

$$\text{“row multiply"} = \begin{bmatrix} 1 & ... & ... & ... \\ ... & 1 & ... & ... \\ \vdots & \vdots & \alpha_i & \vdots \\ ... & ... & ... & 1 \end{bmatrix}$$

$$\text{“row swap"} = \begin{bmatrix} 1 & ... & ... & ... \\ ... & 0 & 1 & ... \\ ... & 1 & 0 & ... \\ ... & ... & ... & 1 \end{bmatrix}$$

We can make these systematic. Let's call the matrix

$$\begin{bmatrix} l_{11} & 0 & ... & 0 \\ l_{21} & l_{22} & ... & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & ... & l_{nn} \end{bmatrix} = I - \tau e_i^T \quad where \quad \tau = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \tau_{i+1} \\ \vdots \\ \tau_n \end{bmatrix}, e^i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8.6}$$

Note that the output product $(UV^T)_{ij} = U_i V_j$.
Here's the idea. We want to upper-triangularize $A \in R^{n \times n}$.

$$\begin{bmatrix} 1 & ... & ... & ... \\ -\tau_1^{(1)} & 1 & ... & ... \\ \vdots & \vdots & \vdots & \vdots \\ -\tau_n^{(1)} & ... & ... & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & ... & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & ... & a_{2n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & ... & a_{nn}^{(1)} \end{bmatrix} \tag{8.7}$$

Exercise 6 Figure out what each of $\tau_i^{(1)}$ needs to be.

$$\begin{bmatrix} 1 & ... & ... & ... \\ 0 & 1 & ... & ... \\ 0 & -\tau_2^{(2)} & ... & ... \\ \vdots & \vdots & \vdots & \vdots \\ 0 & -\tau_n^{(2)} & ... & 1 \end{bmatrix} \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & ... & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & ... & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & ... & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & ... & a_{nn}^{(1)} \end{bmatrix} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & ... & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & ... & a_{2n}^{(2)} \\ 0 & 0 & ... & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & ... & a_{nn}^{(2)} \end{bmatrix} \tag{8.8}$$

If we label the k^{th} Gauss transform M_k where $M_K = I - \tau^k e_k^T$, then we have:

$$M_{n-1} M_{n-2} ... M_1 A = U \tag{8.9}$$

Exercise 7 Show that $M_k^{-1} = I + \tau^k e_k^T$.

We can rewrite the equation as

$$A = M_1^{-1} ... M_{n-1}^{-1} U \tag{8.10}$$

Exercise 8 Prove that $M_1^{-1} ... M_{n-1}^{-1}$ is a lower triangular matrix L.

Comments:
1) The number $a_{kk}^{(k)}$ is called a pivot. It turns out that:

$$det(A_{1:k,1:k}) \neq 0 \iff a_{kk}^{(k)} \neq 0$$

2)

$$(I + \tau^{(1)} e_1^T) ... (I + \tau^{(n-1)} e_{n-1}^T) = \begin{bmatrix} 1 & ... & ... & 0 \\ \tau_1^{(1)} & 1 & ... & ... \\ \tau_2^{(1)} & \tau_2^{(2)} & ... & ... \\ \vdots & \vdots & \vdots & \vdots \\ \tau_n^{(1)} & \tau_n^{(2)} & ... & 1 \end{bmatrix} \tag{8.11}$$

Chapter 9

Solving System of Linear Equations

[SFP: Scribed by Maosen Tang.]

9.1 Block LDU decomposition

Let's consider computing a factorization of a 2×2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{9.1}$$

And we want to perform a LDU decomposition on it assuming that I don't know that $p^{-1}q = qp^{-1}$ if $p, q \in \mathbb{R}$, and $p \neq 0$. (Remember $AB \neq BA$)

Start with eliminating c, we expand A into the expression below:

$$A = \begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \begin{pmatrix} a & b \\ 0 & d - ca^{-1}b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & d - ca^{-1}b \end{pmatrix} \begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix} \tag{9.2}$$

Now, we tried to take the inverse of matrix A:

$$\begin{aligned} A^{-1} &= \left(\begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & d - ca^{-1}b \end{pmatrix} \begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix} \right)^{-1} \\ &= \left(\begin{pmatrix} 1 & a^{-1}b \\ 0 & 1 \end{pmatrix} \right)^{-1} \left(\begin{pmatrix} a & 0 \\ 0 & d - ca^{-1}b \end{pmatrix} \right)^{-1} \left(\begin{pmatrix} 1 & 0 \\ ca^{-1} & 1 \end{pmatrix} \right)^{-1} \\ &= \begin{pmatrix} 1 & -a^{-1}b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a^{-1} & 0 \\ 0 & (d - ca^{-1}b)^{-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -ca^{-1} & 1 \end{pmatrix} \end{aligned}$$

However, in order for this trick to work we need to have $d - ca^{-1}b \neq 0$. Also, since Determinant$[A] = ad - cb \implies \frac{1}{a} det[A] = d - ca^{-1}b$ if $a_{ii} \neq 0 \ \forall i$, it is sufficient to say that if determinant of a matrix is not zero, we could take its inverse through LDU decomposition. Moreover, we can further deduce that $inverse(A) = \frac{1}{det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

9.2 Block Matrices

A block matrix is just a matrix whose element are "compatibly" sized matrices. Example:

$$\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & \ddots & \vdots \\ A_{p1} & \cdots & A_{pq} \end{pmatrix} \tag{9.3}$$

If $A_{ij} \in \mathbb{R}^{m_i \times n_j}$ where $m_i > 0, n_j > 0 \; \forall i, j$, then $A \in \mathbb{R}^{M \times N}$ where $M = m_1 + m_2 + ... + m_p$, and $N = n_1 + n_2 + ... + n_q$. Example (more explicit):

$$\left(\begin{array}{cc|ccc} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ \hline A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} \end{array}\right)$$

The above 6×5 matrix is partitioned into four matrix as the graph instructed, where $m_1 = 2, m_2 = 4, n_1 = 2, n_2 = 3$.

Now let's us consider the matrix $A \in \mathbb{R}^{n \times n}$, let's partion A and rewrite the partitioned matrix and then apply LDU decomposition on it:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix} \begin{pmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{pmatrix} \tag{9.4}$$

We take the inverse of A like what we did in the previous section so we obtain:

$$A^{-1} = \begin{pmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{pmatrix} \begin{pmatrix} A_{11}^{-1} & 0 \\ 0 & (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{pmatrix} \tag{9.5}$$

We then apply this method recursively on each of the smaller matrix: For example, we will do $A_{11} = L_{11}D_{11}U_{11}$

The Matrix $A_{22} - A_{21}A_{11}^{-1}A_{12}$ is called a Schur complement, it is very important in block Gaussian elimination. Since we can perform LDU decomposition on it: $A_{22} - A_{21}A_{11}^{-1}A_{12} = L_{22}D_{22}U_{22}$

Exercise: describe how you can create a recursive algorithm for computing $A = LDU$, where L is unit lower triangular and U is unit upper triangular and D is diagonal (actually diagonal). To make it easy, assume n (number: $A \in \mathbb{R}^{n \times n}$) is a power of two. Hint: what is the recursive base case (n=1)?

Also, you are encouraged to read Chapter 3 of "Matrix Composition" by Golub and Von Loan, in it, you will find: 1.A block LU algorithm which is recursive 2.And also an iterative version

9.3 Programming HW (hint)

Implement LU decomposing with partial pivoting. Example:

$$A = \begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 10000 & 1 \end{pmatrix} \begin{pmatrix} 0.0001 & 1 \\ 0 & -9999 \end{pmatrix} \tag{9.6}$$

PERMUTATION MATRICES: a matrix $P \in \mathbb{R}^{n \times n}$, $P_{ij} \in 0, 1 \; \forall i, j$ and for each i there is exactly one j s.t. $P_{ij} = 1$ and the j is different for each i.

Example: Let $P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, then:

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} c \\ a \\ b \end{pmatrix} \tag{9.7}$$

and

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} c \\ a \\ b \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{9.8}$$

Note that:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = P^T \begin{pmatrix} c \\ a \\ b \end{pmatrix} = P^T P \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{9.9}$$

We could obtain that $P^T = P^{-1}$

p.s. If P and Q are permutation, then $P \cdot Q$ is a permutation.

Let's say $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, so if I write $PA = LU$ (complete the LU decomposition of PA)

$$PA = \begin{pmatrix} 1 & 1 \\ 0.0001 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0.0001 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 0.9999 \end{pmatrix} \tag{9.10}$$

(PA: permute rows of A, AP: permute columns of A)

LU decomposition "algorithm" is like: $A = L_1 A^1 = L_1 L_2 A^2 = L_1 L_2 L_3 A^3$... And we keep performing this step till we recieve a triangular matrix. To avoid roundoff errors that could be caused when dividing every entry of a row by a pivot value that is relatively small in comparison to its remaining row entries. We could do partial pivoting as below:

Partial Pivoting:

$$A^1 = L_1 P_1 A$$
$$A^2 = L_2 P_2 A^1 = L_2 P_2 L_1 P_1 A$$
$$...$$

Exercise: Read Golub and Von Loan: Partial pivoting LU.

9.4 Vector Norm:

if $u, v \in \mathbb{R}^n$, then $||\cdot|| : \mathbb{R}^n \to \mathbb{R}$ is a vector norm if:
(1)$||u+v|| \leq ||u|| + ||v||$ (Triangular inequality)
(2)$||\alpha \cdot u|| = |\alpha| \cdot ||u|| \; \forall \alpha \in \mathbb{R}$ (Absolute Homogeneity)
(3)$u = 0 \iff ||u|| = 0$ (positive-definiteness)
Example: Vector Norm:

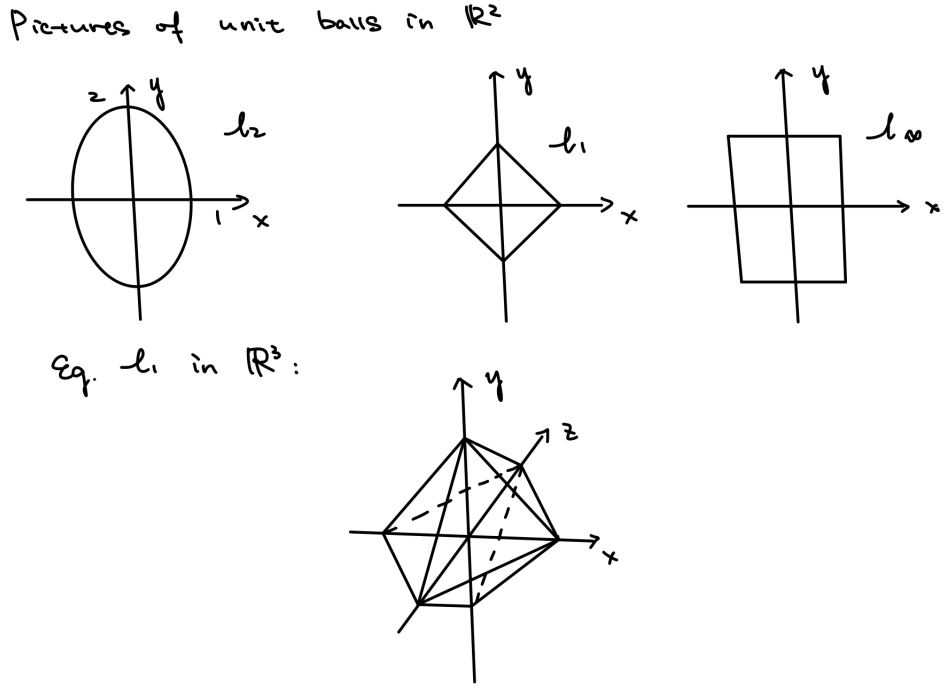
l_2 norm: $||u||_2 = \sqrt{u_1^2 + ... + u_n^2}$ (Recall: $||u-v||_2$ is the euclidean distance)
l_1 norm: $||u||_1 = |u_1| + ... + |u_n|$
l_∞ norm: $||u||_\infty = max_{1 \leq i \leq n} |u_i|$
l_p norm: $(\Sigma_{i=1}^n |u_i|^p)^{\frac{1}{p}}$

9.5 Matrix Norm:

$||\cdot|| : \mathbb{R}^{m \times n} \to \mathbb{R}$ is a matrix norm if: (1)-(3) hold and:
$||AB|| \leq ||A|| \cdot ||B||$ (sub-multiplicity)
Defined: induced (l_p) matrices norms $||A||_p$ is

$$||A||_p = max_{u \neq 0} \frac{||Au||_p}{||u||_p} = max_{||u||_p = 1} ||Au||_p \tag{9.11}$$

Note: the set $\{u \in \mathbb{R}^n : ||u||_p = 1\}$ is the unit ball for the l_p norm on $\mathbb{R}^n$
Example: p=2, n=2:$\{u \in \mathbb{R}^2 | \; ||u||_2 = 1\} = \{x, y \in \mathbb{R} | \sqrt{x^2 + y^2} = 1\}$ (unit circle)



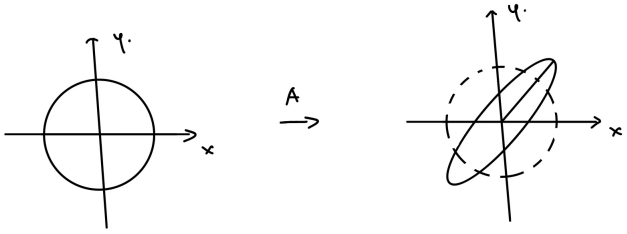
In the special cases of p=1, ∞ the induced matrix norms can be computed or estimated by:

$$||A||_1 = max_{1\leq j\leq n}||A_{ij}||_1 \tag{9.12}$$

and

$$||A||_\infty = max_{1\leq i\leq m}||A_{i,:}||_1 \tag{9.13}$$

However for the case $p = 2$ we need to do more works:
Theorem: $||A||_2 = \sqrt{\lambda_{max}(A^T A)}$



proof: let $u \in \mathbb{S}^{n-1} \leftarrow$ unit hypersphere in $\mathbb{R}^n$ i.e. $||u||_2 = 1$

$$||Au||_2^2 = (Au, Au) = (u, A^T Au) \tag{9.14}$$

Where $A^T A$ is symmetric positive semi-definite then

$$||Au||_2^2 = (u, Q\Lambda Q^T u) = (Q^T u, \Lambda Q^T u) \tag{9.15}$$

($A^T A = Q\Lambda Q^T u$) which is the eigenvalue decomposition where $\Lambda_{ij} = \lambda_i(A^T A) \geq 0\ \forall i$

Note Q is an orthogonal matrix $\implies Q^T Q = 1\ ||Qx||_2 = ||x||_2$ or $Qx^1 \implies \exists v : ||v|| = 1$ ($v = Q^T u$) s.t. $(Q^T u, \Lambda Q^T u) = (v, \Lambda v)$ So,

$$||A||_2^2 = max_{||v||_2=1}||A_u||_2^2 = max_{||v||_2=1}(v, \Lambda v) = max_{1\leq i\leq n}\Lambda_{ij} = \lambda(A^T A) \tag{9.16}$$

Chapter 10

Least squares and the Cholesky decomposition

Over-determined systems

$$\mathrm{Ax} = \mathrm{b}$$
$$\mathrm{A} \in \mathbb{R}^{mxn}$$
$$\mathrm{x} \in \mathbb{R}^{n}$$
$$\mathrm{b} \in \mathbb{R}^{m}$$

Where $m > n$: More equations than unknowns.

How do we solve? And where does this come up?

Eg. Fitting a line with some noise.

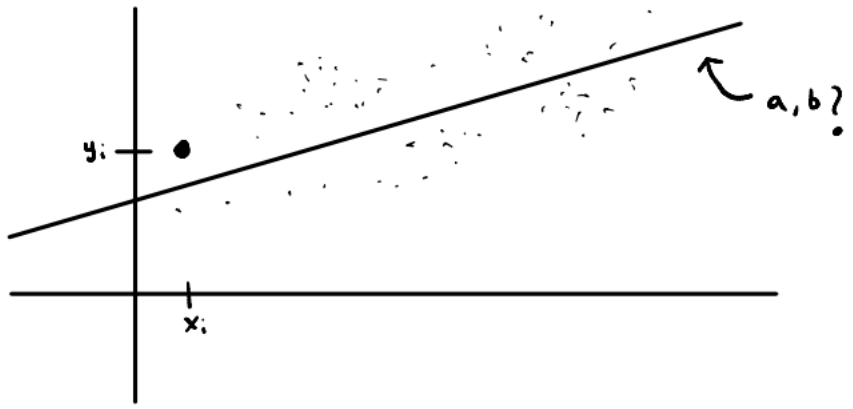
$$y_i = f(x_i) + e_i,\ 1 \leq i \leq m$$

And assume:

$$f(x) = ax + b$$

Goal: estimate a and b.

Picture:



Assume the functional dependence $y_i = f(x_i)$

After fitting, we will have some error:

$$y_i - f(x_i) = e_i$$

In matrix form we can write:

$$\begin{matrix} y_0 \\ y_1 \\ y_0' \\ y_1' \end{matrix} = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 1 & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \\ \epsilon_m \end{bmatrix} \tag{10.1}$$

For $n = 2$.

We could write down $Ax \neq y$.

How do we find "good" x (or (a,b))?

$$\text{Minimize: } \frac{1}{m} \sum_{n=1}^{\infty} 2^{-n} = 1 \text{ w.r.t x = (a,b)}$$

Or:

Minimizing the "Mean square error". If I solve this minimization problem:

$$min_{(a,b)} \frac{1}{m} \sum_{i=1}^{m} \xi^2 = min_{(a,b)} \frac{1}{m} \sum_{i=1}^{m} (y_i - ax_i - b)^2$$

Take partials derivativs w.r.t a and b, xt equal to zero....

Exercise:

Show that you get

$$a = \frac{\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{m}(x_i-\bar{x})^2}$$

$b = \bar{y} - a\bar{x}$, where $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} v_i$, and the same goes for $\bar{y}$

To model this problem correctly, a statistical would make some assumptions about the error distribution. So, if we assume that the errors are 2d Gaussian random variables.

$$\xi_i \sim N(0, \sigma_i^2)$$

and we do a MLE, we get that minimization problem.

Separation:

Solving over determined linear systems:

What about general A$\in \mathbb{R}^{mxn}, m > n$
Let's forget the variable "ξ" and use:

$$r = b - Ax$$

r is called the "residual".

So, to solve (remember: in general, we can't expect Ax=b to hold because it is over-determined).

Minimize sum of squared errors:

$$\sum_{i=1}^{m} r_i^2 = r \cdot r = r^T r = \|r\|_2^2 = \|r\|^2$$

(If I ever don't include the subscript on a norm, assume $\|.\|$)

$$\begin{aligned} r_i &= (b - Ax)_i \\ &= b_i - \sum_{j=1}^{n} A_{ij} x_j \end{aligned}$$

Take the partial derivative w.r.t x_k:

$$\begin{aligned} \frac{\partial x_i}{\partial x_k} &= \frac{\partial d}{\partial x_k} \sum_{j=1}^{n} A_{ij} x_j \\ &= b_i - \sum_{j=1}^{n} A_{ij} x_j \\ &= \sum_{j=1}^{m} A_{ij} \frac{\partial x_j}{\partial x_k} \end{aligned}$$

$$= \sum_{j=1}^{n} A_{ij} \frac{\partial x_j}{\partial x_k}$$
$$= \sum_{j=1}^{n} A_i j \delta_{jk}$$
$$\text{where } \delta_{jk} == \begin{cases} 1 & \text{if } j < k \\ 0 & \text{else} \end{cases}$$

This is the Kronecker delta.

So:

$$\frac{\delta}{\delta x_k} \sum_{i=1}^{m} r_i^2 = 2 \sum_{i=1}^{m} r_i \frac{\delta r_i}{\delta x_k} = 2 \sum_{i=1}^{m} A_{ik} r_i$$
$$= 2 \sum_{i=1}^{m} (a_k) r_i$$
$$= 2 a_k^T r$$
$$= 2 a_k^T (b - Ax)$$

Remember:
Goal is to minimize $\sum_{i=1}^{m} r_i^2 wrtx$.

Recall:
"first -order necessary conditions for optimally" are:

→ gradient of the function to be minimized must be equal to zero.

In our case we need:

$$e(x) = \sum_{i=1}^{m} r_i^2, \text{ need } 0 = \nabla e = (\frac{\delta e}{\delta x_i}, ..., \frac{\delta e}{\delta x_n})$$
$$= 2((a_1^T (b - A_x), ..., a_n^T (b - A_x))$$
$$= 2A^T (b - Ax)$$

So we need:

$$A^T (b - Ax) = 0$$

These are called the "normal equations". Those are the equations you get if you solve an over-determined system and assume the residuals are 2d normal.

$$\boxed{\mathrm{A}^t Ax = A^t b}$$

We could we have gotten to the normal equations with less woke? Yes.
Remember the definition of a derivative:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In terms of Taylor expansions, we want this:

$$f(x+h) = f(x) + f'(x)h + O(h^2)$$

We have:

$$e(x) = \|r(x)\|^2$$

$$(x+h) = b - A(x+h) = b - Ax - Ah$$

$$\frac{\delta r}{\delta x} = -A$$

Next we also have the following relations:

$$\begin{aligned} e(x) &= \nabla ||r(x)||^2 \nabla \\ &= \nabla r(x) \cdot r(x) \\ &= 2\nabla r(x)^T r(x) \\ &= -2\nabla A^T r(x) \\ &= -2A^T(b - Ax) \end{aligned}$$

Exercise:
Compute $\nabla e(x)$ using the "Taylor expansion trick" directly.

have the NE's and some idea that solving them $\Longleftrightarrow$ solving LS problem, which is statistically ok.

When can we solve:

$A^T Ax = A^T b$
$A^T A \to mxn$

Operation	Cost
$LU = A^T A$	$O(n^3)$
$A^T b$	$O(mn)$
$L^{-1} A^T b$	$O(n^2)$
$U^{-1} L^{-1} A^T b$	$O(n^2)$

Figure 10.1: Caption

$x \to n$
$A^T b \to n$
INSERT IMAGE

Solvable if A has full column rank.

Exercise:
What needs to be true of the line fitting option for its A to be full rank?

How do we solve it?

$$A^t A x = A^t b$$
$$x = (A^T A)^{-1} A^T b$$
$$= A^T b$$

The matrix $A^\dagger$ is called the Moore-Penrose pseudo inverse. Can't compute this unless you have to ...

Instead, e.g.

$A^T A = LU \to LUx = A^T b$
$\to Ux = L^{-1} A^T b \to x = U^{-1} L^{-1} A^T b$

Note: for small n, i.e. n=O(1) w.r.t m this is only O(m)

Cholesky decomposition
More LA:

Definition: a matrix A is positive definite if for all $x \neq 0$, $x^T Ax > 0$

Remember, for A$\in \mathbb{R}^{mxn}$, $A = A^T$ if λ_n are the eigenvalues of A, then $\lambda_i \in \mathbb{R} \forall_i$

Key property:

$$A = A^T and x^T Ax > 0 \forall x \neq 0$$

$\Rightarrow$ all positive (real) eigenvalues.

Such a matrix is called symmetric positive definite, abbreviate it spd.

Why should we expect this to be true? Assume A is full rank.

$A = Q \wedge Q^T, where\ Q = [q_1, ..., q_n]\ and\ q_i^T q_j = S_{ij}$

$$S_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{else} \end{cases} \tag{10.2}$$

i.e. $q_{i\,i=1}^{n}$ forms an orthonormal basis for $\mathbb{R}^n$.

Another way is to write this:

$$A = \sum_{i=1}^{n}$$

Exercise:
Prove this ($Q \wedge Q = \sum_{i=1}^{n}$)

The fact that q_i's are eigenvalues. not important. this is just linear algebra indexing practice.

Let $x \neq 0$ then

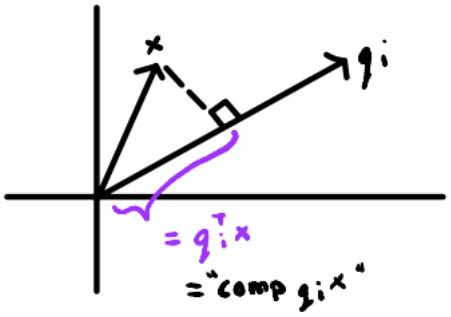
$$x^T Ax = \sum_{i=1}^{m} \lambda_i x^T q_i q_i^x x = \sum_{i=1}^{n} \lambda_i (q_i^T x)^2$$

Well, we have assumed $x \neq 0 \Rightarrow x^T Ax > 0$. But assume for now that for some i st $1 \leq i \leq \lambda$ that $\lambda_i \leq 0$. Well .... $q_i \neq 0$.
But:

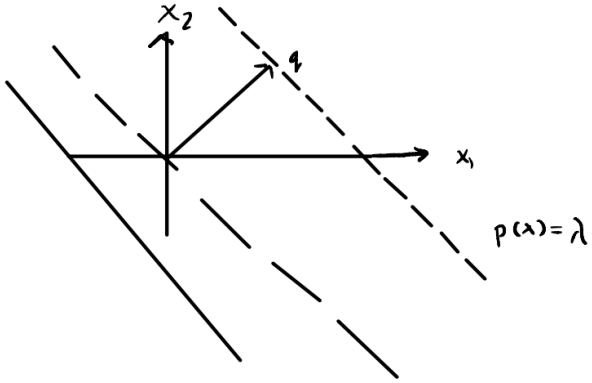
$$q_i^T Aq_i = \sum_{j=1}^{n} \lambda_j (q_j^T q_i)^2 = \lambda_i \leq 0$$

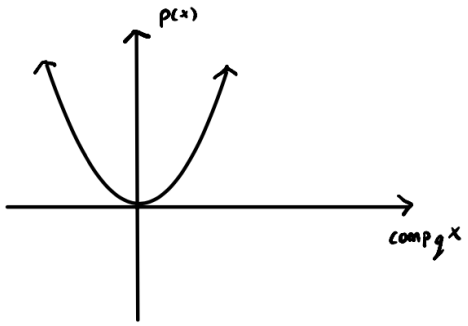
Contradiction.
Note: $q_i^T x = comp_{qi} x$

Picture



(If: $\|q_i\| = 1$ Otherwise: $comp_n x = \frac{u^T x}{\|u\|}$





Exercise:

So we have a "quadratic form" $p(x) = x^T Ax$, we used the eig decomp $A = Q \wedge Q^T$, where $Q = [q_1, ...q_n], \wedge = diag(\lambda_i)$, to decompose p(x) into functions like $p_i(x) = \lambda_i (comp_{q_i} x)^2$ acting orthogonal directions. Show how to do this for

$$P(x) = x^T Ax + b^T x + c$$

Constants: A, b, c

A$\in \mathbb{R}^{nxn}$

b$\in \mathbb{R}^{nx1}$

c$\in \mathbb{R}^{1x1}$

pos	def	$X^T A x > 0$	$\lambda_i > 0 \; \forall i$
pos	Semi def	$x^T A x \geq 0$	$\lambda_i \geq 0 \; \forall i$
neg	def	$x^T A x < 0$	$\lambda_i < 0 \; \forall i$
neg	def	$x^T A x \leq 0$	$\lambda_i \leq 0 \; \forall i$
indefinite		$x^T A x < 0$	$\exists i,j \; . \; \lambda_i > 0, \lambda_j < 0$

Chapter 11

QR Factorization and Gram-Schmidt Orthogonalization

[SFP: Scribed by Junyao Chen.]

11.1 Recap from last lecture

Summary of last lecture:

1. Motivated least squares (e.g fit a line to many noisy observations). Get overdetermined system:
$$Ax = b, \quad A \in \mathbb{R}^{m \times n}$$
Think of m as # of observations and n as # of parameters.

2. Discussed where "least squares" comes from: minimized mean square error (MSE). Statistically sound. Assume errors ("residuals") are normally distributed.

3. Based on this, derived the normal equations:
$$A^T Ax = A^T b$$

4. If A is full (column) rank, then A^T is full rank and can solve. (Note: $(A^T A)^{-1} A^T$ is called "pseudoinverse".)

5. We can now solve the normal equations using e.g. LU decomposition.

6. But we can do better. We can use a "Cholesky" decomposition: $A^T A = R^T R$, where R is upper trianglular. ($R = L^T$)

7. To prove that this is possible, we need to take a detour and learn about "positive definiteness".

8. The key to understanding (positive) definiteness of a symmetric matrix such as A^TA is to look at its eigenvalue decomposition, $Q\Lambda Q^T$. Here $Q = [q_1, \cdots, q_n]$ is the matrix of orthogonal eigenvectors and $\Lambda = diag(\lambda_i)_{i=1}^n$ is the diagonal matrix of eigenvalues. They will all be real since A^TA is a symmetric matrix. Their sign characterizs definiteness.

9. It is helpful to relate x^TAx to a type of polynomial called "quadratic form". A quadratic form is a type of quadratic function associated to a symmetric matrix. It's basically an n-dim version of $f(x) = ax^2$. This is easy to do using $Q\Lambda Q^T$.

11.2 Lemma

(Very critical, powerful, far-reaching lemma)
Lemma: Let $A \in \mathbb{R}^{m\times m}$ be spd (symmetric positive definite matrix). Let $x \in \mathbb{R}^{m\times n}$ where $m \geq n$ be full rank (column rank because $m \geq n$). Then x^TAx is spd.

Proof. 1. sym: $(x^TAx)^T = x^TA^T(x^T)^T = x^TA^Tx = x^TAx$.

2. pd: Recall definition of pd (positive definiteness): Let $z \neq 0$. Then since x has full column rank, $xz \neq 0$. So let $y = xz \neq 0$. Then

$$z^Tx^TAxz = y^TAy > 0$$

because A is pd.

□

Exercise: Let A be pd. Then for all i, $a_{ii} > 0$. *Hint: use previous lemma*. It's easy to prove but may get stuck when first time seeing it.

11.3 Cholesky Decomposition

Theorem (Cholesky): Let A be spd. Then there exists a unique lower triangular matrix L s.t. $L_{ii} > 0$ and $A = LL^T$.

Note: If L is lower triangular ($L = [\blacktriangleleft]$) , the L^T is upper triangle: $L^T = [\blacktriangleleft]^T = [\blacktriangleright]$, and vice versa.

This theorem is just algorithms. Programming it is the recommended way to learn this.

Proof. If $A \in \mathbb{R}^{n\times m}$, then $a_{11} \in \mathbb{R}, w_1 \in \mathbb{R}^{(n-1)\times 1}, k_1 \in \mathbb{R}^{(n-1)\times(n-1)}$.

Start by blocking A like:

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & w_i^T \\ w_1 & k_1 \end{bmatrix} \\ &= \begin{bmatrix} \alpha_1 & 0 \\ \frac{w_1}{\alpha_1} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & k_1 - w_1 \end{bmatrix} \begin{bmatrix} \alpha_1 & \frac{w_1^T}{\alpha_1} \\ 0 & I \end{bmatrix} \\ &= L_1A_1L_1^T \end{aligned}$$

□

Notice that by the lemma above, A is more than pd now: A is spd.

We know that $a_{11} > 0$, so $\alpha_1 = \sqrt{a_{11}} > 0$. Also notice that each matrix (L_1, A_1, L_1^T) is invertible because $det(L_1) = det(L_1^T) = \alpha_1 \neq 0$ and $det(A_1) = k_1 - w_1 \neq 0$.

So write $A_1 = L_1^{-1} A L_1^{-T}$. (Notation clarification: $L^{-T} = (L^T)^{-1} = (L^{-1})^T$).

Since L_1 is invertible, L_1 is full rank. So L^{-T} has full rank. Now apply the lemma with $x = L_1^{-T}$ (i.e L_1^{-T} plays the role of x). Since A is spd, we conclude that A_1 is spd.

Exercise: Use lemma to prove that A_1 spd implies $k_1 - \frac{w_1 w_1^T}{d}$ is spd. *Hint: Solution is very similar to previous exercise.Recursively apply this argument to* $k_1 - \frac{w_1 w_1^T}{d}$. *Then done.* (Cholesky).

Exercise: Show that $A^T A$ is positive <u>semi</u>-definite ($x \neq 0 \implies x^T A x \geq 0$). And A is positive definite if A is full (column) rank.

Side remark: $A \in R^{m \times n}$, m: # observations, n: # parameters. $A = \begin{bmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{bmatrix}$, so a_i is a vector of m observations of parameter i.

So if A is **<u>NOT</u>** full rank, then there must exist a pair at columns a_i, a_j s.t $\exists \alpha \neq 0$: $a_i = \alpha a_j$.

If we compute the Cholesky decompositions, then

$$A^T A x = A^T b \text{ (normal equation) } \rightarrow LL^T x = A^T b \text{ (easy to solve)}$$

Furthermore, compared to LU, computing Choleky takes about $1/2$ as many FLOPs.

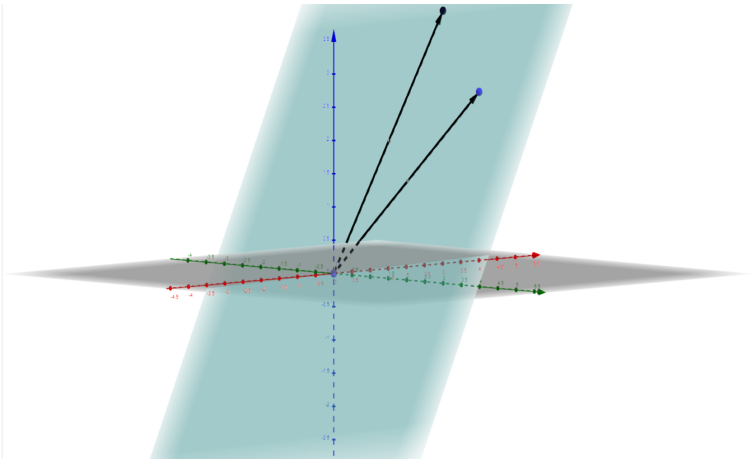
Note: no need for pivoting. Since pd, each pivot > 0. Pivoting is overhead, meaning lots of data swapping. So if data satisfy Cholesky condition, then we can solve more efficiently.

There are a few more ways to solve the normal equations.

11.4 QR Factorization

Now think of A (defined in side remarks) "geometrically".

<u>E.g</u> $A \in \mathbb{R}^{3 \times 2}$. So $A = [a_1, a_2]$ where $a_1, a_2 \in \mathbb{R}^3$. So $Ax = x_1 a_1 + x_2 a_2$. If a_1 and a_2 are linearly independent, so that rank$(A) = 2$, then $\{a_1, a_2\}$ span a 2D linear subspace of $\mathbb{R}^3$ (i.e a plane). A is overdetermined. Use the following picture as guide.



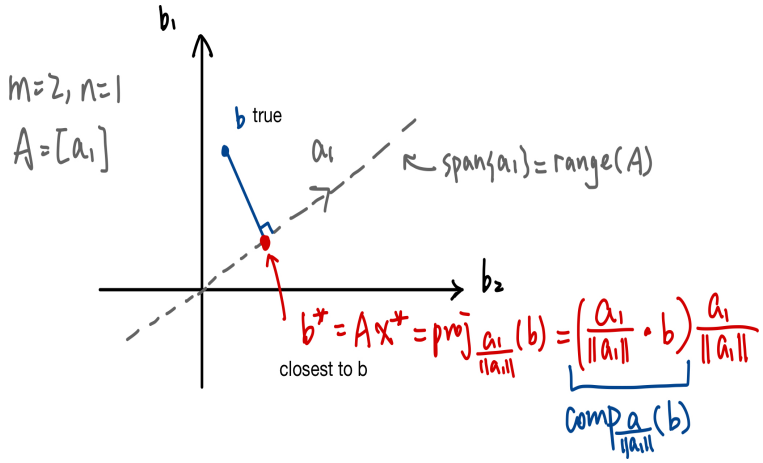
Two vectors span a plane in 3D
Source: GeoGebra Visualization

Four vectors in ten dimensions meaning we don't have enough vectors to span the whole space. That's why we cannot solve the explicit solution. To find a sensible solution here, the one we've learned is to minimized the sum of squares.

In general, there may be no vector of coefficients "x") such that b can be expressed as a linear combination of columns of A. I.e there does not exist x such that $Ax = x_1a_1 + \cdots + x_na_n = b$. That is, the point b doesn't lie on the plane (span of $\{a_1, \cdots a_n\}$). We've decided to minimize $\|r\|^2 = \|b - Ax\|^2$

Goal: find x^* such that $\|b - Ax\|^2$ is minimized. **Then**, Ax^* is the closest point in span $\{a_1, ..., a_n\}$ to b.

A picture for the case where $m = 2$, $n = 1$, $A = \{a_1\}$ is illustrated below.



Least square minimization when $m = 2$, $n = 1$, $A = \{a_1\}$

In the picture, the closest point b^* to the true (target) value b is

$$b^* = \text{proj}_{\frac{a_1}{\|a_1\|}}(b) = \left(\frac{a_1}{\|a_1\|} \cdot b\right)\frac{a_1}{\|a_1\|} = \text{comp}_{\frac{a_1}{\|a_1\|}}(b)\frac{a_1}{\|a_1\|}$$

Notice that the important geometric properties remain the same for higher dimensions. If A is full rank and square, then $Ax = b \Longrightarrow x = A^{-1}b$.

At the same time, if $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, then $Q^{-1} = Q^T$. So $x = Q^T b$ where $Q = [q_1, ..., q_n]$.

$$I = Q^{-1}Q = Q^T Q = \begin{bmatrix} q_1^T q_1 & \cdots & q_1^T q_n| \\ \vdots & \ddots & \vdots \\ q_n^T q_1 & \cdots & q_n^T q_n \end{bmatrix}$$

An intermediate case: Let Q be "semi-orthogonal". So $Q \in \mathbb{R}^{m\times n}$, $m > n$ and $Q^T Q = I_n$. In this case, the overdetermined system is easy to solve: $Qx = b$ gives $x = Q^T b = \begin{bmatrix} q_1^T b \\ \vdots \\ q_n^T b \end{bmatrix}$

Idea: start with linear independent $\{a_1, \cdots, a_n\}$, and run $Gram - Schmidt$, which turns the basis into orthogonal basis, so then the equations are easy to solve.

Result: $A = QR$ where $Q \in \mathbb{R}^{m\times n}$ is semi-orthogonal (so $Q^T Q = I$ but note $QQ^T \neq I$) and R is upper triangular.

Remark: $(A^T A)^{-1}A^T = A^\dagger$ (psedoinverse of A) is the matrix version of "comp_a" And $A^\dagger$ is like proj_a.

So $A(A^T A)^{-1}A^T$ projects a vector into the column space of A. Note: if Q is semi-orthogonal, then $QQ^\dagger = Q(Q^T Q)^{-1}Q^T = QIQ^T = QQ^T$. So now notice that $Q^\dagger = Q^T$

11.5 Gram-Schmidt Orthogonalization

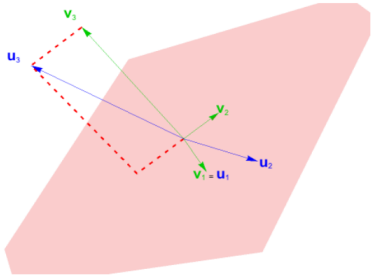
Start with the columns of A, i.e $\{a_1, \cdots, a_n\}$, $a_i \in \mathbb{R}^m$, $m \geq n$. (In the HW, we'll look at $m < n$).

Goal: convert $\{a_1, \cdots, a_n\}$ into $\{q_1, \cdots, q_n\}$ such that $\text{span}\{a_1, \cdots, a_n\} = \text{span}\{q_1, \cdots, q_n\}$, and $q_i^T q_j = \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{else} \end{cases}$

Algorithm 4 Gram-Schmidt Orthogonalization for 2D with 3 vectors

set $\tilde{q_1} = a_1$
set $q_1 = \frac{\tilde{q_1}}{\|\tilde{q_1}\|}$
$\tilde{q_1} = a_2 - (q_1 \cdot a_2)q_1$
$q_2 = \frac{\tilde{q_2}}{\|\tilde{q_2}\|}$
$\tilde{q_3} = a_3 - q_1 q_1^T a_3 - q_2 q_2^T a_3 = (I - q_1 q_1^T - q_2 q_2^T)a_3 = (I - QQ^T)a_3$, where $Q = [q_1, q_2]$
$q_3 = \frac{\tilde{q_3}}{\|\tilde{q_3}\|}$

Notice that q_1 and q_2 are orthogonal:
$q_1^T q_2 = q_1^T(a_2 - q_1 q_1^T a_2) = q_1^T a_2 - q_1^T q_1 q_1^T a_2 = q_1^T a_2 - q_1^T a_2 = 0$
Remember: $A = QR$, $Q^T Q = I$, and R is upper triangluar.



Gram-Schmidt orthogonalization in 3D (Relate $a_i = u_i$ and $q_i = v_i$)
Source: Wolfram Demonstration Project

Algorithm 5 (Classic) Gram-Schmidt Orthogonalization

for $j = 1, \cdots, n$ **do**
  $\tilde{q}_j = a_j$
  for $i = 1, \cdots, j-1$ **do**
    $r_{ij} = q_i^T a_j$
    $\tilde{q}_j = q_j - r_{ij} q_i$
  end for
  $r_{jj} = \|\tilde{q}_j\|$
  $q_j = \frac{\tilde{q}_j}{r_{jj}}$
end for

How to solve overdetermined system using QR?
Answer: Compute $A = QR$. Then solve $x = R^{-1} Q^T b$
Unfortunately G-S orthogonalization is numerically unstable: Due to numerical round-off error, we may get extremely inaccurate resul. The floating point errors build up fast.
Instead , we want to use one of the better alternatives: *Modified Gram-Schmidt*
Observation: We could write $\tilde{q}_3 = a_3 - q_1 q_1^T a_3 - q_2 q_2^T a_3$. But this is equal to

$$\begin{aligned}\tilde{q}_3 &= (I - q_1 q_1^T)(I - q_2 q_2^T) a_3 \\ &= (I - q_1 q_1^T - q_2 q_2^T + q_1 q_1^T - q_2 q_2^T) a_3 \\ &= (I - q_1 q_1^T - q_2 q_2^T + 0) a_3\end{aligned}$$

Key idea: "$I - q_i q_i^T$" is a matrix which, when multiplied, does "q_i rejection". (Extends to $I - QQ^T$)

Algorithm 6 Modified Gram-Schmidt Orthogonalization

for $i = 1, \cdots, n$ **do**
  $\tilde{q}_i = a_i$
end for
for $i = 1, \cdots, n$ **do**
  $r_{ii} = \|\tilde{q}_i\|$
  $q_i = \frac{q_i}{r_{ii}} \rightarrow$ *"numerically stable", much more accurate*
  for $j = i + 1, \cdots, n$ **do**
    $r_{ij} = q_i^T \tilde{q}_j$
    $\tilde{q}_j = \tilde{q}_j - r_{ij} q_i$
  end for
end for

Chapter 12

QR Householder

[SFP: Scribed by Andy Luo]

Previously: Gram-Schmidt, Modified G-S (More Numerically Stable)

$\{a_1, ..., a_n\}$ orthogonalize $\rightarrow \{q_1, ..., q_n\}$

$(a_i, q_i \in \mathbb{R}^m \forall \mathbb{R}^m)$

$\rightarrow$ A=QR, $Q^TQ = I$, $Q \in \mathbb{R}^{m \times n}$

$QQ^T \neq I$ necessarily (So Q is semi-orthogonal and R is upper triangular)

For this to work, we need A to be full column rank, in which case we solve the Least Square problem or the over-determined :

$$Ax = b$$

Two other main tools for QR:

- Householder reflections
- Givens Rotations

12.1 Triangle Orthogonalization

$$A \rightarrow AR_1 = \hat{Q}_1$$
$$\rightarrow AR_1R_2 = \hat{Q}_2$$
$$\vdots$$
$$\rightarrow AR_1...R_n = Q$$
$$\rightarrow A = QR, R = R_n^{-1}...R_1^{-1}$$

We are operating on the column space of A. What about working on the row space of A?

12.2 "Orthogonal Triangularization"

$$
\begin{aligned}
A &\to Q_1 A = \hat{R}_1 \\
&\to Q_2 Q_1 A = \hat{R}_2 \\
&\vdots \\
&\to Q_n \dots Q_1 A = R \\
&QA = R
\end{aligned}
$$

- Need Q to be orthogonal
- Need R to be upper triangular

Note:

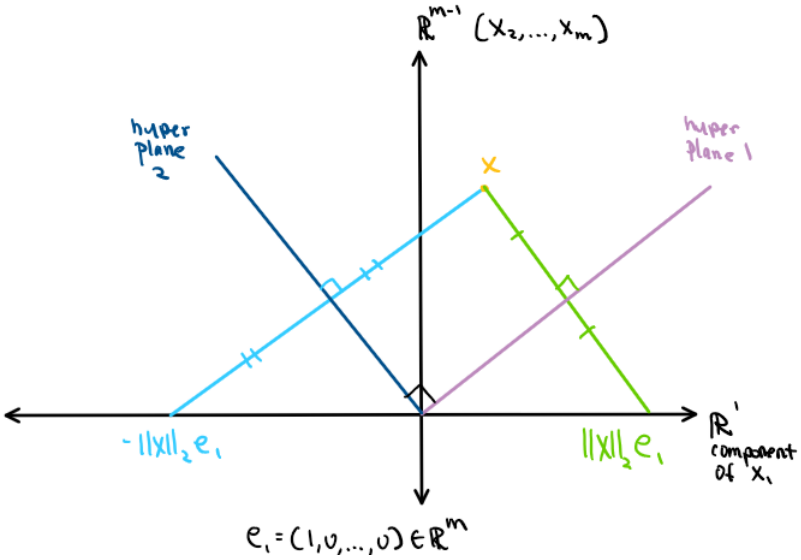
$$\text{If } Q_i^T Q_i = I \ \forall i \text{ Then}$$

$$Q^T Q = (Q_n \dots Q_1)^T (Q_n \dots Q_1) = I$$

How do we do this?

$$
\underset{A}{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}} \to \underset{Q_1 A}{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{bmatrix}} \to \underset{Q_2 Q_1 A}{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & * \end{bmatrix}} \to \underset{\begin{matrix} Q_3 Q_2 Q_1 A \\ = QA \\ = R \end{matrix}}{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \\ 0 & 0 & 0 \end{bmatrix}}
$$

Picture of orthogonal transformation



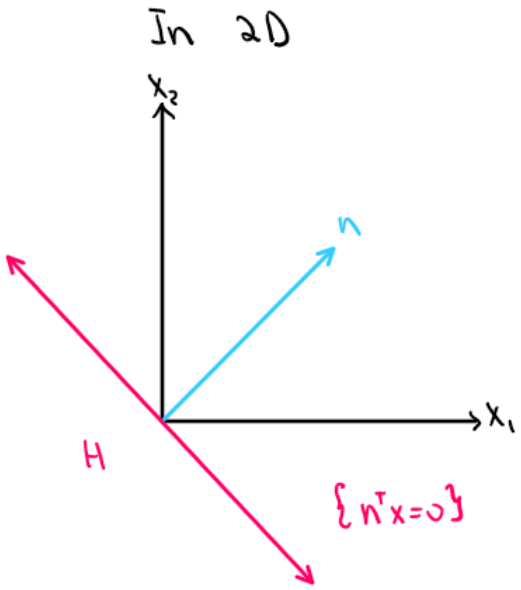
Observations:

1. x and $\pm||x||_2 e_1$ are related by an orthogonal transformation
2. Let $Q^TQ = I$, then if $y = Qx$ then $||y||_2^2 = y^Ty = x^TQ^TQx = x^Tx = ||x||_2^2$

Recall:
If $n \in \mathbb{R}^m$ is a unit vector ($n^Tn = 1, ||n||_2 = 1$), then n defines a hyperplane in $\mathbb{R}^m$ by:

$$H = \{x \in \mathbb{R}^m : n^Tx = 0\}$$

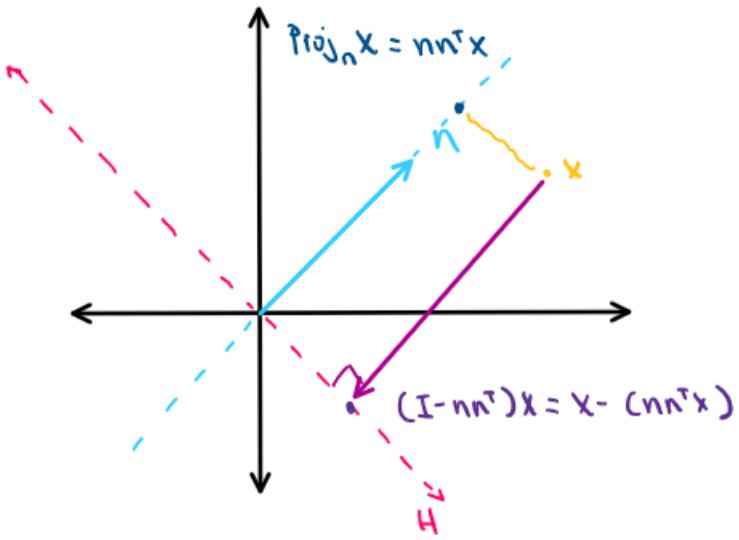
and the dimension of H is m-1.



12.2.1 Project a point onto a Hyperplane

Recall:

$$proj_n x = (n^T x) \cdot n = nn^T x$$



We have the "vector rejection" $x - (n^T x)n = (I - nn^T)x$

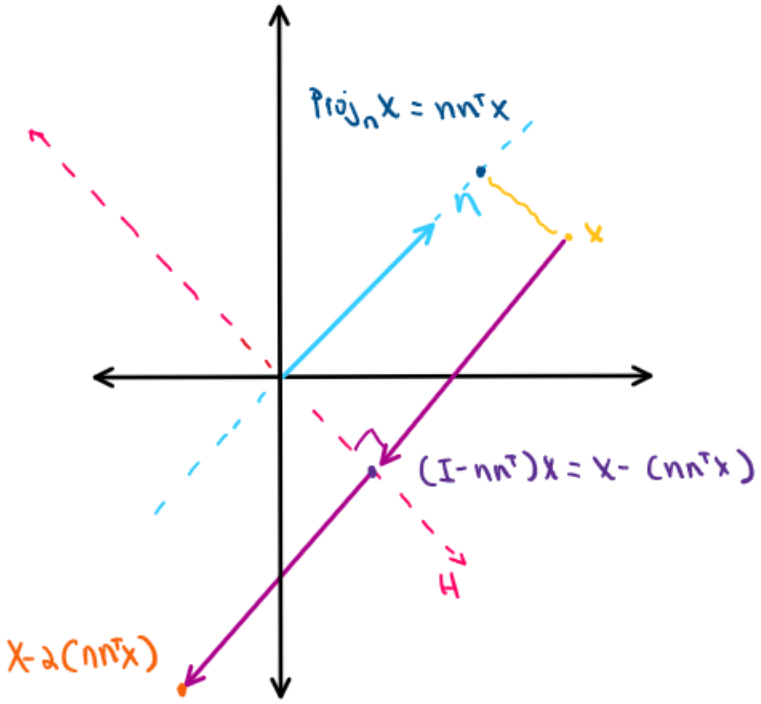
$(I - nn^T)x$ is the orthogonal projection of x onto the hyperplane derived by n that is

$$H = \{x \in \mathbb{R}^m; n^T x = 0\}$$

Check:

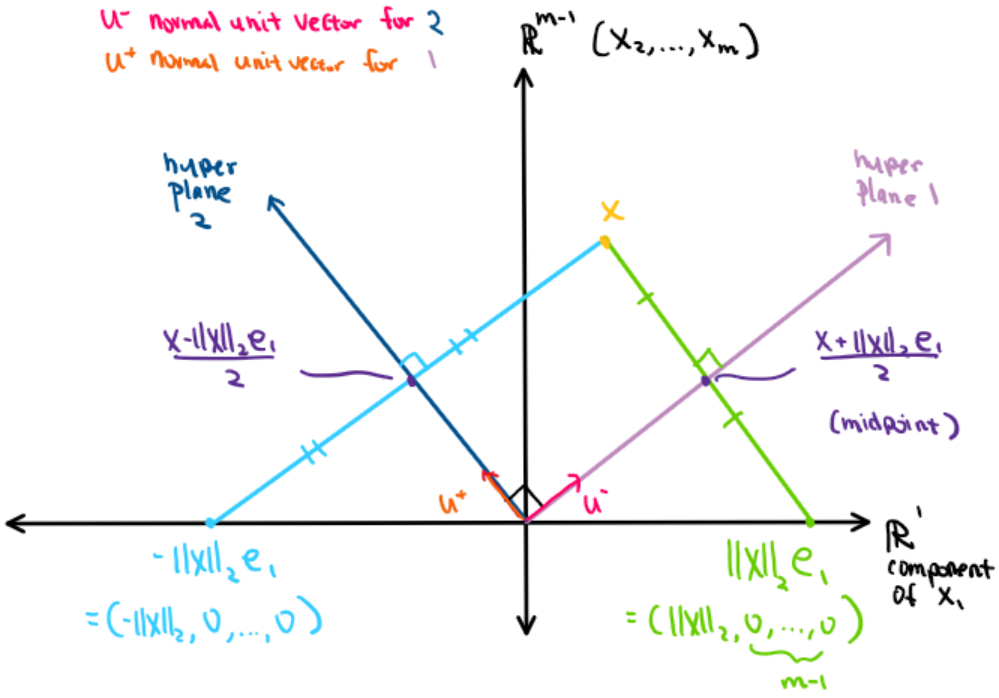
$$\begin{aligned} & n^T(I - nn^T)x \\ &= n^T x - n^T nn^t x \\ &= n^T x - ||n||_2^2 n^T x \\ &= n^T x - n^T x = 0 \end{aligned}$$

12.2.2 Reflect a point over a plane



Do a double vector rejection $(I - 2nn^T)x$.

12.3 Back to QR Decomposition



$$u_+ = \frac{x - ||x||_2 e_1}{||x - ||x||_2 e_1||_2}$$

$$u_- = \frac{x + ||x||_2 e_1}{||x + ||x||_2 e_1||_2}$$

Note: u_+ and u_- are backward because

$$(I - 2u_+ u_+^T)x = ||x||_2 e_1$$

$$(I + 2u_- u_-^T)x = -||x||_2 e_2$$

Remember our first step is to choose Q_1 such that

$$\underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{A} \rightarrow \underbrace{\begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{bmatrix}}_{Q_1 A}$$

Just pick $Q_1 = I - 2u_\pm u_\pm^T$ where $u_\pm = \frac{a_1 \mp ||a_1||_2 e_1}{||a_1 \mp ||a_1||_2 e_1||_2}$

We want to pick $\pm$ such that "x" after being reflected has moved the largest distance. This is to avoid numerical instability due to rounding error accumulation.

In particular, we want to avoid catastrophic cancellation.

12.4 Catastrophic Cancellation

Example Joe is 159.49 cm and Bob is 159.51 cm. The true difference is 0.02cm. However, if the measurement material can only be accurate to the 1cm. The approximate difference then would be 1 cm. This is 50 times larger !

Relative Error Subtraction is numerically illed-conditioned. This is not intrinsic to floating point arithmetic. It's just that if our subtract approximate quantities are closed to each other, you can lose a lot of relative accuracy.

Let $\hat{x} = x(1 + \mathcal{E}_x)$ and $\hat{y} = y(1 + \mathcal{E}_y)$
Then $\hat{x} - \hat{y} = x - y + x\mathcal{E}_x - y\mathcal{E}_y$
$= (x - y) + \frac{x-y}{x-y}(x\mathcal{E}_x - y\mathcal{E}_y)$
$= (x - y)(\frac{1+x\mathcal{E}_x - y\mathcal{E}_y}{x-y})$
$= (x - y)(\mathcal{E}_{x-y})$

Note: $\mathcal{E}_{x-y}$ can be arbitrary large if x-y (the denominator) is small

12.5 Algorithm for Householder QR: $A \in \mathbb{R}^{m \times n}$

---

for $k = 1, ..., n$: **do**
  $x = A_{K:m,k}$
  $u_k = x \pm ||x||_2 e_1$ {select sign base on largest distance}
  $u_k = \frac{u_k}{||u_k||_2}$
  $A_{k:m,k:n} = (I - 2u_k u_k^T) \cdot A_{k:m,k:n}$
end for

Note: After, A is overwritten with R and we have only indirectly computed Q. We don't Have Q, just actions.

Sidebar: Don't need to store all minimum number in matrix to multiply with it. Some examples:

- def multiply_with_I(x):

    return x

  O(1) instead of O(mn)

- def multiply_w_reflection(n,x):

    return x-(n@x)n

  @ is the dot product, O(m) space, O(m) time

12.6 Column Pivoted QR

Let's say, for $A \in \mathbb{R}^{m \times n}$, r=rank(A) and r<min(m,n). So A is rank deficient. If we have a QR decomposition of A. Then:

$$det(A) = 0 = det(QR) = det(Q)det(R)$$

Recall det(Q)=1 because Q is orthogonal.

$$det(R) = 0$$

Recall R is an upper triangle and thus

$$r_{11}r_{22}...r_{nn} = 0$$

At least one of the r values on the diagonal is 0. Thus the algorithm for Household runs into a problem!

Idea: Want a QR factorization algorithm that is okay with rank deficient matrices.

Assume for $k \geq 1$ that:

$$H_{k-1}H_{k-2}...H_1AP_1P_2...P_{k-1} = \begin{bmatrix} R_{11}^{(k-1)} & R_{12}^{(k-1)} \\ 0 & R_{22}^{(k-1)} \end{bmatrix}$$

Where the H matrices are the householder reflectors and the P matrices are the column swap permutation matrices.

$R_{11}^{(k-1)} \in \mathbb{R}^{(k-1)\times(k-1)}$
$R_{12}^{(k-1)} \in \mathbb{R}^{(k-1)\times(n-k+1)}$

These 2 block matrices are good in the column pivoting. Look at $R_{22}^{(k-1)}$.

$R_{22}^{(k-1)} \in \mathbb{R}^{(m-k+1)\times(n-k+1)}$
$=[z_1, z_2, ..., z_{n-k+1}]$

Such that$z_j \in \mathbb{R}^{m-k+1}$

Want to find z_j such that $\|z_j\|$ is maximize and then apply permutation $\hat{P}_k$ on $R_{22}^{(k-1)}$ base on this and choose H_k to be upper triangular.

$$R_{22}^{(k-1)}\hat{P}_k$$

To apply to the original matrix, the permutation P_k would be

$$P_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & \hat{P}_k \end{bmatrix}$$

Where I_{k-1} is of dimension $\mathbb{R}^{(k-1)\times(k-1)}$ and $\hat{P}_k \in \mathbb{R}^{(n-k+1)\times(n-k+1)}$.

12.7 Problems

- Explain why choosing $u_{\pm}$ so that x moves the larger distance helps with catastrophic cancellation.
- Compute the eigenvalues and determinant of

$$I - 2nn^T$$

  where $||n||_2 = 1$, HINT: Use Geometry.
- Prove that if a matrix Q is orthogonal then det(Q)=1.

Chapter 13

Singular Value Decomposition(SVD)

[SFP: Scribed by Allen Wang]

"The most important matrix factorization/decomposition"

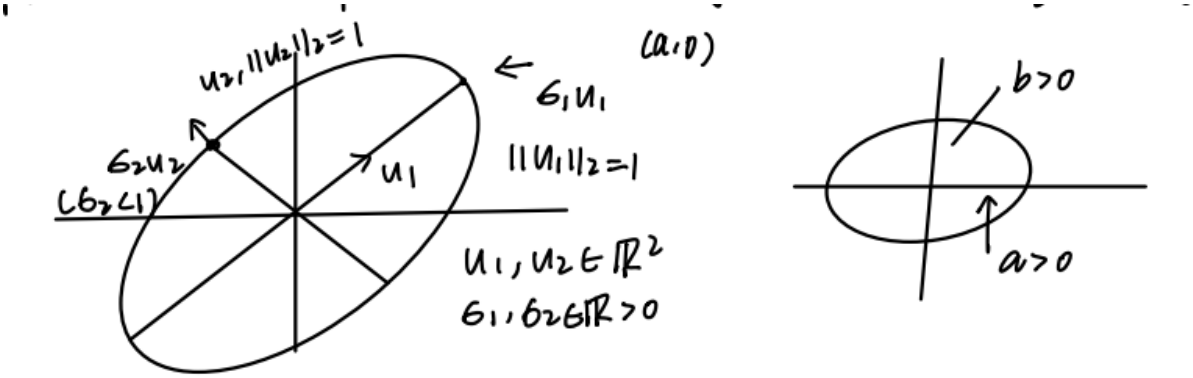
1. Every matrix $A \in C^{m \times n}$ has a "unique" SVD
2. The SVD tells us everything about a matrix

What is it?

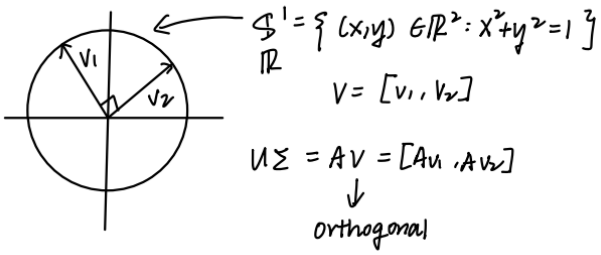
13.1 Geometric Fact

Let $A \in C^{m \times n}$ and let $S^{n-1} = \{u \in C^n : \|u\|_2 = 1\}$. So, S^{n-1} is the $n-1$ dimensional unit hypershpere in C^n. Then, the image of S^{n-1} under A is a hyperellipsoid. So: the set $\{A_n : u \in S^{n-1}\}$ is a "hyperellipsoid".

Reminder: an ellipse in R^2 : $\{(x, y) \in R^2 : \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1\}$



$$\begin{bmatrix}\sigma_1 u_1 & \sigma_2 u_2\end{bmatrix} = \begin{bmatrix}u_1 & u_2\end{bmatrix} \begin{bmatrix}\sigma_1 & \\ & \sigma_2\end{bmatrix} = u \sum, A \in R^{2 \times 2}$$



Since v is orthogonal $vv^T = I$. So, $A = u \sum v^T$. This is the SVD of A.

Theoremm

Let $A \in C^{m \times n}$. Then, the SVD $A = u \sum v^*$ ($*$ is conjugate transpose) is unique. These matrices have those properties:

1. $u = \begin{bmatrix} u_1 & ... & u_r \end{bmatrix} \in C^{m \times r}$ is the matrix of left singular vectors where $r = rank(A) \leq min(m, n)$. It is (semi-) orthogonal: $u^* u = I$. Note: if $r < m$, then $uu^* \neq I$.
2. $v = \begin{bmatrix} v_1 & ... & v_r \end{bmatrix} \in C^{n \times r}$ is the matrix whose columns are the right singular vectors. Same thing: $v^* v = I$ and $vv^* \neq I$ unless $n = r$.
3. $\sum = \begin{bmatrix} \sigma_1 & ... & 0 \\ 0 & \ddots & 0 \\ 0 & ... & \sigma_r \end{bmatrix} \in R^{r \times r}$ is the diagonal matrix of singular values where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r$

13.2 Proof of SVD

First, the idea:

1. we'll isolate the "direction of the largest action of A"
2. we'll "eliminate" this direction and induction the dimension

Set:

$$\sigma_1 = \|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2 \rightarrow \text{largest singular}$$

(Note: since $\{\|x\|_2 = 1\} = S^{n-1}$ is a compact set (closed and bounded), this maximum exists)

$$v_1 = \arg\max_{\|r\|_2 = 1} \|Ar\| \rightarrow \text{maximum argument v attains maximum exists } x^* \text{ s.t } f(x^*)$$

$$u_1 = \frac{Av_1}{\|Av_1\|} = \frac{Av_1}{\sigma_1} \rightarrow \sigma_1 u_1 = Av_1$$

Define the "orthonormal extensions" for u_1 and v_1 to get the matrices:

$$u_1 = \begin{bmatrix} u_1 & \widetilde{u}_2 & \dots & \widetilde{u}_m \end{bmatrix} \in C^{m\times m} \quad v_1 = \begin{bmatrix} v_1 & \widetilde{v}_2 & \dots & \widetilde{v}_n \end{bmatrix} \in C^{n\times n}$$

To do this, just pick arbitrary unit vectors. (To make $u_1^* u_1 = u_1 u_1^* = I$, $v_1^* v_1 = v_1 v_1^* = I$)

Then:
($A = u \sum v^*$, $u^* A v = u^* u \sum v^* v = \sum$)

$$u_1^* A v_1 = \begin{bmatrix} u_1^* \\ \widetilde{u}_2^* \\ \vdots \\ \widetilde{u}_m^* \end{bmatrix} A \begin{bmatrix} v_1 & \widetilde{v}_2 & \dots & \widetilde{v}_n \end{bmatrix} = \begin{bmatrix} u_1^* A v_1 & u_1^* A \widetilde{v}_2 & \dots & u_1^* A \widetilde{v}_n \\ \widetilde{u}_2^* A \widetilde{v}_1 & \widetilde{u}_2^* A \widetilde{v}_2 & \dots & \widetilde{u}_2^* A \widetilde{v}_n \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{u}_m^* A \widetilde{v}_1 & \widetilde{u}_m^* A \widetilde{v}_2 & \dots & \widetilde{u}_m^* A \widetilde{v}_n \end{bmatrix} = \begin{bmatrix} \sigma_1 u_1^* u_1 & * & \dots & * \\ \sigma_1 \widetilde{u}_2^* u_1 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_1 \widetilde{u}_m^* u_1 & * & \dots & * \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * \end{bmatrix} = \begin{bmatrix} \sigma_1 & w^* \\ O_{(m-1)\times 1} & B \end{bmatrix} = S_1$$

$w^* \in C^{n-1\times 1}$.
Sidebar: $x \in R^n$, $y \in R^n$, $\alpha \in R$, $x^T(\alpha y) = \alpha x^T y$

Exercise: Note $S_1 = u_1^* A v_1$ show that $\|S_1\|_2 = \sigma_1$
Observe that:

$$\left\| S_1 \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \sigma_1 & w^* \\ O & B \end{bmatrix} \begin{bmatrix} \sigma_1 & w \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \sigma_1^2 + w^* w \\ Bw \end{bmatrix} \right\|_2 = \sqrt{(\sigma_1^2 + w^* w)^2 + \|Bw\|_2^2}$$

$$\geq \sqrt{(\sigma_1^2 + w^* w)^2} = \sigma_1^2 + w^* w$$

Notice:

$$\sigma_1^2 + w^* w = \sqrt{\sigma_1^2 + w^* w} \cdot \sqrt{\sigma_1^2 + \|w\|_2^2} = \sqrt{\sigma_1^2 + w^* w} \left\| \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2$$

Hence: setting $v = \begin{bmatrix} \sigma_1 \\ w \end{bmatrix}$

$$\sigma_1 = \|S_1\|_2 = \max_{v \neq 0} \frac{\|s_1 v\|_2}{\|v\|_2} \geq \frac{\left\| s_1 \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2}{\left\| \begin{bmatrix} \sigma_1 \\ w \end{bmatrix} \right\|_2} \geq \sqrt{\sigma_1^2 + \|w\|_2^2} \geq 0$$

Square both sides to get:
$\sigma_1^2 \geq \sigma_1^2 + \|w\|_2^2 \geq 0$

Subtract σ_1^2 from both sides:
$0 \geq \|w\|_2^2 \geq 0 \rightarrow \|w\|_2^2 = 0 \rightarrow w = 0$

$$u_1^* A v_1 = s_1 = \begin{bmatrix} \sigma_1 & w^* \\ 0 & B \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix}$$

We're basically done.

Exercise:

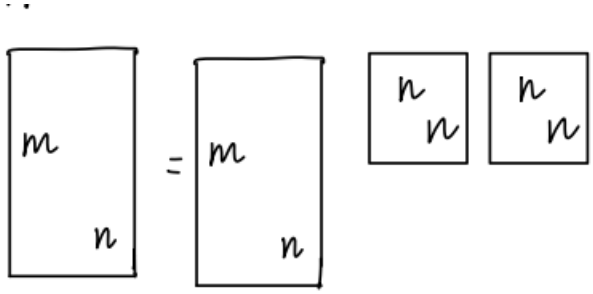
Apply this same idea to B and shows that at the end, we can write $u_2^* A v_2 = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & B_2 \end{bmatrix}$ where $u_2 = \begin{bmatrix} u_1 & u_2 & \widetilde{u}_3 & \dots & \widetilde{u}_m \end{bmatrix}$ $v_2 = \begin{bmatrix} v_1 & v_2 & \widetilde{v}_3 & \dots & \widetilde{v}_n \end{bmatrix}$

Exercise:

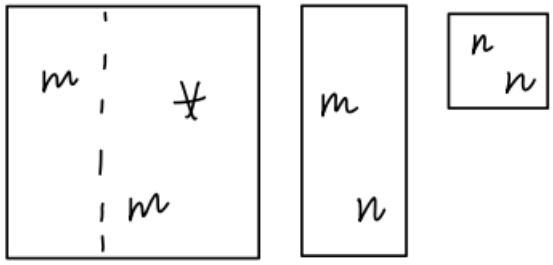
What happens if $\arg\max_{\|r\|_2=1} \|Av\|$ is not unique?

13.3 Reduced SVD vs. Full SVD

reduced:



full:



13.4 Properties of the SVD

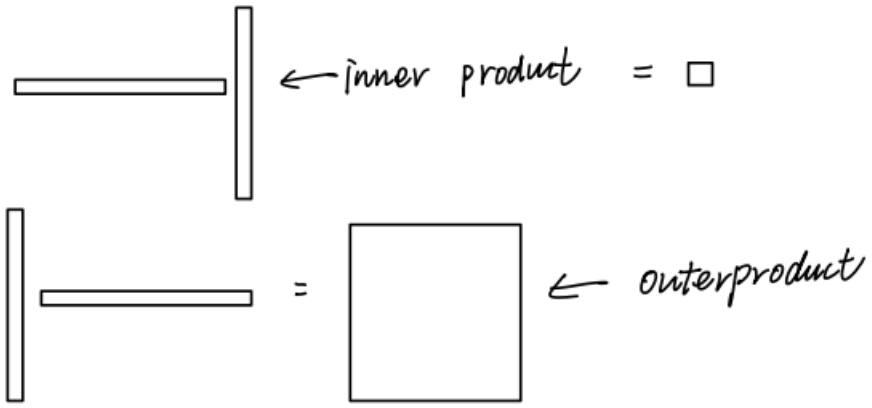
Exercise:

1. rank(A) = r = # nonzero $\sigma_i's$
2. range(A) = $\langle u_1, \quad ... \quad , u_r \rangle$
3. $\|A\|_2 = \sigma_1$, $\|A\|_F = \sum_{i=1}^{r} \sigma_i^2$ $\quad (\|A\|_F = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2)$
4. nonzero σ_i's for A are the square root of the nonzero λ_i's of $A^T A$ or AA^T
5. if $A = A^T$, then the σ_i's are just $|\lambda_i| \, of \, A$
6. $|det(A)| = \prod_{i=1}^{n} \sigma_i$ if A is square

13.5 Low Rank Approximation

Let $A \in C^{m \times r}$ and let $B \in C^{m \times p}$, $C \in C^{p \times n}$. Then, if $\|A - BC\|_2$ is small and $p << \min(m, n)$, then BC is a "low rank approximation" of A.

Exercise:
If $A = u \sum v^*$, the $A = \sum_{i=1}^{r} \sigma_i u_i v_i^*$ (writing A as a sum of outer products)



Exercise:
Show that if $u \in R^m$ and $v \in R^n$, then $rank(uv^T) = 1$

Theorem:
Let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^*$, $k \leq r = rank(A)$. If $k + 1 > \min(m, n)$, set $\sigma_{k+1} = \delta$. Then $\|A - A_k\|_2 = \inf_{rank(B) \leq K} \|A - B\|_2 = \sigma_{k+1}$ where $B \in C^{m \times n}$

Note: A_k is called a "truncated" singular decomposition. What the Thm says is the k-term truncated SVD is the best rank k (low rank) approximation in the matrix 2-norm sense.

Chapter 14

Eigenvalue Algorithms

[SFP: Scribed by Lucas Hsu]

14.1 Announcements

PHW#3

- Finished writing
- Kind of long
- Due Sunday, 4/10

The professor is having a baby soon, so send emails!

14.2 Eigenvalues I: The power Method

This is the last week of numerical linear algebra, with eigenvalues. Then we will move on to other topics, such as multivariate root finding, interpolation, and quadratic differential equations.

Quick Review: The eigenvalue problem. For $A \in \mathbb{R}^{n\times n}$, find a scalar $\lambda \in \mathbb{C}$ and a vector $u \in \mathbb{R}^n$ such that $Au = \lambda u$. This is true when $\det(A - \lambda I) = 0$, or by finding the solutions λ to $p(\lambda)$, the characteristic polynomial.

Recall: By the Fundamental Theorem of Algebra, $p(\lambda) = (z_1 - \lambda_1)\dots(z_n - \lambda_n)$, which implies that $\exists n$ λ_i's. Also, if $n \geq 5$, then the degree $p(\lambda) \geq 5$ as well. Then, we do not have a closed form solution to the eigenvalue problem in general. This is a result of Galois Theory, written by Evariste Galois, who died in a duel at age 21.
To amuse yourself, you might want to look up the cubic formula and the quartic formula, the analogs of the quadratic formula, but for polynomials of those degrees.

Let's say that we have n eigenpairs of (λ_i, u_i) such that $< u_1, \dots, u_n >$, or the span of the eigenvectors (the linear combinations of the eigenvectors), is equal to $\mathbb{R}^n$. This is the same as saying that $\{u_i's\}$ form a basis. Then we can stack the individual eigenvalue problems "$Au_i = \lambda_i u_i \varepsilon$ into matrix form to get:

$$AU = U\Lambda$$

where $U = [u_1 \dots u_n] \in \mathbb{R}^{n \times n}$, and

$$\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n) = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \in \mathbb{R}^{n \times n}$$

Then, since the u_i's are linearly independent ($< u_1, \dots, u_n >= \mathbb{R}^n$), we have that $U = [u_1, \dots, u_n]$ is full column rank, and as such U is invertible, or we have U^{-1} such that $UU^{-1} = U^{-1}U = I$. So, we use the equation above and multiply on the right by U^{-1} and find:

$$A = U\Lambda U^{-1}$$

Then, A is diagonalizable. Note that if the λ_i's are all distinct, then A is diagonalizable. However, this is is only sufficient, not necessary.

Theorem 1 (The Spectral Theorem for Symmetric Matrices)**.** *If $A \in \mathbb{R}^{n \times n}$ is symmetric, then there is an eigendecomposition of A:*

$$A = Q\Lambda Q^T$$

where Q is orthogonal and $\lambda_i \in \mathbb{R}$.

Note: if A is also positive definite, then λ_i's are positive.

14.2.1 Applications

Solving Systems of ODEs
Let's say we have $A \in \mathbb{R}^{n \times n}$, which is diagonalizable.
Let $x(t) = (x_i(t), \dots, x_n(t))$, a curve in $\mathbb{R}^n$. As such, the function x is $x : \mathbb{R} \to \mathbb{R}^n$, a curve or trajectory in $\mathbb{R}^n$.
Let us also assume that $\dot{x}(t) = (\dot{x}_1(t), \dots, \dot{x}_n(t))$, where $\dot{x}_1(t) = \frac{dx_1}{dt}(t)$, and that this satisfies $\dot{x}(t) = Ax(t)$, and that $x(0) = x_0$, our initial condition.

To solve, let us write $A = U\Lambda U^{-1}$, the eigenvalue decomposition. Then, define $y(t) = U^{-1}x(t)$.
A useful way of thinking: if $A = [a_1 \dots a_n]$, then multiplication by A^{-1} has the effect of expressing a vector in the basis $\{a_i\}_{i=1}^n$. The u_i's are "modes". As such, multiplying $U^{-1}x(t)$ finds the coefficients of $x(t)$ in the set of U modes.
Then we see that

$$\dot{y}(t) = \frac{d}{dt}y(t) = \frac{d}{dt}U^{-1}x(t) = U^{-1}(\dot{t})$$

And so:

$$\dot{x}(t) = Ax(t) = U\lambda U^{-1}x(t) = U\Lambda y(t)$$

$$U^{-1}\dot{x}(t) = \Lambda y(t)$$

$$\dot{y}(t) = \Lambda y(t) \rightarrow \begin{cases} \dot{y}_1(t) = \lambda_1 y_1(t) \\ \vdots \\ \dot{y}_n(t) = \lambda_n y_n(t) \end{cases}$$

This has the effect of decoupling the system into n independent scalar ODEs. We can then solve each $\dot{y}_i(t) = \lambda_i y_i(t)$ independently, guessing that $y_i(t) = C_i \exp(\lambda_i t)$, and confirming that $\dot{y}_i(t) = C_i\lambda_i \exp(\lambda_i t) = \lambda_i y_i(t)$. This is quite powerful, as it takes apart the system!

Other Applications

- PHW#3: Spectral Graph Theory
- Page rank: what Google uses to index the internet. To read about this, look for SIAM Review's "The $25 billion eigenvector".

14.2.2 Rayleigh Quotient

Here is a useful function to have in mind: the Rayleigh Quotient.
The Rayleigh Quotient: For $A \in \mathbb{R}^{n\times n}$, assuming that $A = A^T$:

$$R(A,u) = \frac{u^T Au}{u^T u} = \frac{u^T Au}{||u||_2||u||_2} = \frac{u^T}{||u||_2} A \frac{u}{||u||_2}$$

Notice that if $v = \alpha u$, where $u \neq 0 \neq v$, we have $R(A,u) = R(A,v)$, so the Rayleigh quotient is the same for all points in a direction. So, it's enough to find $R(A,u)$ on the unit sphere only.
Notice, further, that if $A = Q\Lambda Q^T$, with $Q = [u_1 \dots u_n]$, then for all i,
$R(A,u_i) = \lambda_i$, as:

$$R(A,u_i) = \frac{u_i^T QAQ^T u_i}{u_i^T u_i} = e_i^T \Lambda e_i = \lambda_i$$

(Remember that e_i is the vector with 0's everywhere except in the ith spot, where there is a 1.) Consequently, we have that the minimum and maximum eigenvalues, or $\lambda_{\min} = \min_{1\leq i\leq n} \lambda_i$ and $\lambda_{\max} = \max_{1\leq i\leq n} \lambda_i$, are the boundaries of the Rayleigh quotient. Or:

$$\lambda_{\min} \leq R(A,u) \leq \lambda_{max}$$

Exercise: Prove this!

14.2.3 The Power Method

Idea: notice that:

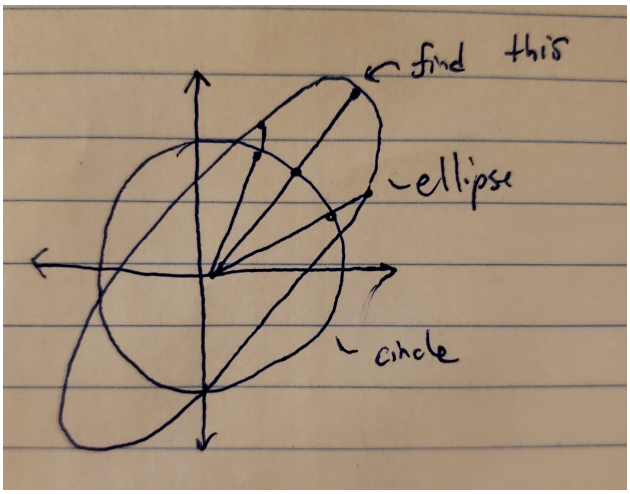
$$Au = \lambda u \rightarrow u = \frac{1}{\lambda} Au$$

This is a fixed point iteration! So, let's say we want a fixed point iteration generating u^k such that $||u^{(k)}||_2 = 1$. The key thing happening in $u^{(k+1)} = \frac{1}{\lambda} Au^{(k)}$ is multiplication by A. Can we just repeatedly multiply by A and normalize $u^(k)$ at each step? Yes!

The Algorithm:

1. Choose $u^(0)$
2. While not converged:
   - Set $u^{(k+1)} = Au^(k)$
   - Set $\lambda^{(k+1)} = R(A, u^{(k+1)}) = \frac{u^{(k+1)T} Au^{(k+1)}}{u^{(k+1)T} u^{(k+1)}}$

Remember that $Au = \lambda u$, so $R(A, u) = \frac{u^T Au}{u^T u} = \lambda_i$. So, if $u \approx u_i$, then $R(A, u) \approx \lambda_i$. So, $u^T Au = \lambda u^T u$.
Remember: a matrix maps the unit sphere to an ellipsoid. In 2D, we're seeking out the direction of the largest deviation in the ellipsoid, like so:



Theorem 2. *Let A be diagonalizable and let A have a "dominant" eigenvalue, so $\lambda_{\max} = \lambda_1$, and we have $\lambda_1 >> \lambda_2$. Then $\exists\, u_c^0$ such that the power method applied to $u_1^{(0)}$ converges to u_1.*

Proof: Let $u_1^{(0)} = u_c = \sum_{i=1}^n e_i u_i$, where $e_i \neq 0$. Let's iterate the power method, or just multiply by A. Then:

$$u_1^{(1)} = Au_1^{(0)} = A \sum_{i=1}^n c_i u_i = U\Lambda U^{-1} Uc = U\Lambda c = \sum_{i=1}^n \lambda_i c_i u_i$$

The effect of multiplying x by A is to scale the ith coefficient of the vector x expressed in the U "eigenbasis" by λ_i.
So, if we do this repeatedly:

$$\begin{aligned} A^k u_1^{(0)} = (U\Lambda U^{-1})^k Uc & \\ &= (U\Lambda U^{-1})(U\Lambda U^{-1}) \dots (U\Lambda U^{-1})Uc \\ &= (U\Lambda^k U^{-1})(Uc) = U\Lambda^k c \\ &= \sum_{i=1}^{n} \lambda_i^k c_i u_i = \lambda_1^k c_1 u_1 + \sum_{i=2}^{n} \lambda_i^k c_i u_i \\ &= \lambda_1^k (c_1 u_1 + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1}\right)^k c_i u_i) \end{aligned}$$

Remember that we assumed that $\lambda_1 > \dots > \lambda_n$, and that λ_1 is dominant, so we know $\lambda_1 >> \lambda_2$, and so $\lambda_1 >> \lambda_i$, $i \geq 2$. Then, as $k \to \infty$, $A^k u_1^{(0)} \to \lambda_1^k c_1 u_1$
How quickly does this happen? It's controlled by the ratio $\frac{\lambda_2}{\lambda_1}$, so if $\lambda_1 = 1.00001$ and $\lambda_2 = 1$, then you multiply the second term by $\frac{1}{1.00001}$ with each iteration, so it is slow to converge.

Exercise: Establish the order and rate of convergence of the power method.

Exercise: What happens if $\lambda_1 = \lambda_2$?

Note: This finds the maximum eigenvalue. To find the minimum eigenvalue, see that $Au = \lambda u \to \frac{1}{\lambda} u = A^{-1}u$. So, the max eigenvalue of A^{-1} is simply $\frac{1}{\lambda_{\min}}$. However, this only works for spd matrices, as otherwise we may have negative eigenvalues.

Example: From PHW#2: we have a matrix $A \in \mathbb{R}^{(N-1)^2 \times (N-1)^2}$. Most rows had a 4 on the diagonal, and 4 -1's somewhere else. So A is a discretized Laplacian matrix. What we did in the homework, maybe without knowing, was doing a finite difference discretization of the partial differental equation: $\Delta u = f$ (the Dirschlet Boundary Condition) on $[-1, 1] \times [-1, 1] \subseteq \mathbb{R}^2$, and $u = 0$ on the boundary of $[-1, 1] \times [-1, 1]$, which is Laplace's equation.
In this case, A is symmetric positive semi-definite. $A = Q\Lambda Q^T$, $\lambda_i \geq 0 \; \forall i$.
But what is "Δ"? It is the differential operator:

$$\Delta = \frac{d^2}{dx_1^2} + \dots + \frac{d^2}{dx_n^2}$$

So, in $\mathbb{R}^1$, $\Delta = \frac{d^2}{dx^2}$. In $\mathbb{R}^2$, $\Delta = \frac{d^2}{dx_1^2} + \frac{d^2}{dx_2^2}$.

But why should we expect A to be spd? Well, we can talk about functions being spd. In $\mathbb{R}^1$, we can show that $\Delta = \frac{d^2}{dx^2}$ is negative semi-definite. In WHW#3, we defined $L^2[a, b]$ to be the set of

square integrable functions on $\mathbb{C}^2[a,b]$: $f \in L^2[a,b] : \int_a^b f(x)^2 dx < \infty$. Then, we defined the inner product $(f,g) = \int_a^b f(x)g(x)dx$, or $x\dot{y} = x^T y$, where $x, y \in \mathbb{R}^n$

Let's show that $\Delta = \frac{d^2}{dx^2}$ is negative symmetric semi-definite:

- Symmetric: assuming that $f, g \in L^2[a,b]$, such that $f(a) = f(b) = 0$ and $g(a) = g(b) = 0$, we want to show that $(\Delta f, g) = (f, \Delta g)$, or that $(Ax)^T y = x^T (Ay)$, which is true if $A = A^T$. We perform Integration By Parts:

$$(\Delta f, g) = \int_a^b f''(x)g(x)dx$$
$$= f'g\Big|_a^b - \int_a^b f'g'dx = -\int_a^b f'g'dx$$
$$= -fg'\Big|_a^b + \int_a^b fg''dx = \int_a^b fg''dx = (f, \Delta g)$$

- Negative Semi-Definite:

$$(\Delta f, f) = \int_a^b f''(x)f(x)dx$$
$$= f'(x)f(x)\Big|_a^b - \int_a^b f'(x)^2 dx$$
$$= -(f', f') = -||f'||_{L^2[a,b]} \leq 0$$

Note: If we expect A to be semi-definite, that means that we have $A = Q\Lambda Q^T$. Then, we can discretize $\Delta u = f$ to get $A\underline{u} = \underline{f}$, where $\underline{u}$ and $\underline{f}$ are vectors. This implies that $Q\Lambda Q^T \underline{u} = \underline{f}$ and so $\Lambda Q^T \underline{u} = Q^T \underline{f}$. We then conclude that we can from $\hat{u} = Q^T \underline{u}$ and $\hat{f} = Q^T \underline{f}$, the spectral coefficients. These, in turn, can lead us to spectral methods and differential equations.

Chapter 15

Eigenvalue Algorithms (Continued)

[SFP: Scribed by Arnav Kanwal]

15.1 Recap from Last Lecture

Eigenvalue Problem:

- Find (λ, u) such that $Au = \lambda u$
- A is diagonizable if there exists a linearly independent basis of eigenvectors $u_1, \ldots, u_n$. Then:

$$A = U\Lambda U^{-1}$$

- Power Method: Iterate $u^{(k+1)} = Au^{(k)}$ and set

$$u^{(k+1)} = \frac{\hat{u}^{(k+1)}}{||\hat{u}^{(k+1)}||}$$

15.2 Finite Difference Discretizations of derivatives

In Programming HW#2, we claimed that a matrix $A \in \mathbb{R}^{(N-1)^2 \times (N-1)^2}$ is a finite difference discretization of the Laplacian operator Δ on $[-1,1] \times [-1,1] \subseteq \mathbb{R}^2$.

We want to solve: $\begin{cases} \Delta u = f & \text{on the interior of the square: } (-1,1) \times (-1,1) \\ u = 0 & \text{on the boundary of } [-1,1] \times [-1,1] \end{cases}$

15.2.1 Explaining the Laplacian operator

Recall:

$$\Delta = \frac{d^2}{dx_1{}^2} + \cdots + \frac{d^2}{dx_n{}^2}$$

$$\text{in } \mathbb{R}^2 : \Delta = \frac{d^2}{dx^2} + \frac{d^2}{dy^2}$$

Δ is a linear operator which means that it sends functions to functions by computing their Laplacian.

Note: It's "linear" because $\Delta(\alpha f + \beta g) = \alpha\Delta f + \beta\Delta g$ where $\alpha, \beta \in \mathbb{R}$ and f, g are functions $\in C^2$. Much like: $(\alpha f + \beta g)'' = \alpha f'' + \beta g''$.

Goal: Discretize Δ to replace it with the matrix $A \in \mathbb{R}^{n\times n}$ which maps vectors $x \in \mathbb{R}^n$ to $y = Ax \in \mathbb{R}^n$ where the vectors x are to be thought of as discretizations of functions.

Sample: Evaluate f at grid points x_i

Simpler Case: Start with the interval $[-1, 1]$, let the integer $n > 0$:

$$x_i = -1 + 2\left(\frac{i}{n}\right), \quad 0 \le i \le n$$

Example: Let $n = 4$. Then $x_0 = -1$, $x_1 = -0.5$, $x_2 = 0$, $x_3 = 0.5$, $x_4 = 1$

15.2.2 Discretizing the derivative

Say we want to discretize $\frac{d}{dx}$. We can do this using Taylor Expansions. Let $h = x_{i+1} - x_i$. Expand:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4)$$

Then we want to subtract the two functions:

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3} + O(h^4)$$

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{h^2}{6}f'''(x) + O(h^3)$$

Now, we can rearrange the equation to isolate $f'(x)$ and get the following:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(x) + O(h^3)$$

Using the derivative equation above, let's say we want to center the equation with grid nodes

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{h^2}{6} f'''(x_i) + O(h^3)$$

We can therefore approximate the derivative as

$$\frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

with error $\frac{h^2}{6} f'''(x_i) + O(h^3)$ (i.e approximation with $O(h^2)$ accuracy).

Let's write this in matrix form

$$\overrightarrow{f} = \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \in \mathbb{R}^{n-1} \qquad \overrightarrow{f'} = \begin{bmatrix} f'(x_0) \\ f'(x_1) \\ \vdots \\ f'(x_n) \end{bmatrix} \in \mathbb{R}^{n+1}$$

Notice, we can pick out the ith element in $\overrightarrow{f'}$

$$\begin{bmatrix} f'(x_0) \\ \vdots \\ f'(x_i) \\ \vdots \\ f'(x_n) \end{bmatrix} = \begin{bmatrix} * \\ \vdots \\ \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \\ \vdots \\ * \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$= \frac{1}{2h} \begin{bmatrix} 0 & \cdots & 0 & -1_{(i,i-1)} & 0_{(i,i)} & 1_{(i,i+1)} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{i-1}) \\ f(x_i) \\ f(x_{i+1}) \\ \vdots \\ f(x_n) \end{bmatrix}$$

$$= \frac{1}{2h} \begin{bmatrix} * & & & \cdots & & & * \\ -1 & 0 & 1 & 0 & \cdots & & 0 \\ 0 & -1 & 0 & 1 & \cdots & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & & \cdots & & -1 & 0 & 1 \\ * & & & \cdots & & & * \end{bmatrix} \begin{bmatrix} f(x_0) \\ f(x_1) \\ \\ \vdots \\ \\ f(x_n) \end{bmatrix}$$

"Finite Discretization Matrix"

15.2.3 Exercises on Discretization and Kronecker Product

Kronecker Product Definition:

Given matrices $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{p \times q}$

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \\ a_{1m}B & \cdots & a_{mn}B \end{bmatrix} \in \mathbb{R}^{mpxnq}$$

Exercise 1:
Figure out the first and last row of the Finite Discretization Matrix we found above (will need a different Taylor Expansion).

Hint: Do Taylor Expansions of $f(x + h)$ and $f(x + 2h)$ for $f'(x_0)$ and TE's of $f(x - h)$ and $f(x - 2h)$ for $f'(x_n)$.

Since x_{-1} and x_{n+1} don't exist in the grid, the goal is to figure out how to cancel terms in the Taylor Expansions such that the error is $O(h^2)$ overall.

Exercise 2:
Do a Finite Difference approximation of $\frac{d^2}{dx^2}$

Note: The Finite Difference approximation of a Laplacian is called a Kronecker Product of $F\lambda$ matrices for $\frac{d^2}{dx^2}$.

$\Rightarrow$ Then in Python, let A be the resulting FD matrix.

1. compute `np.kron(A,A)`
2. compare your result with A from Programming HW#2

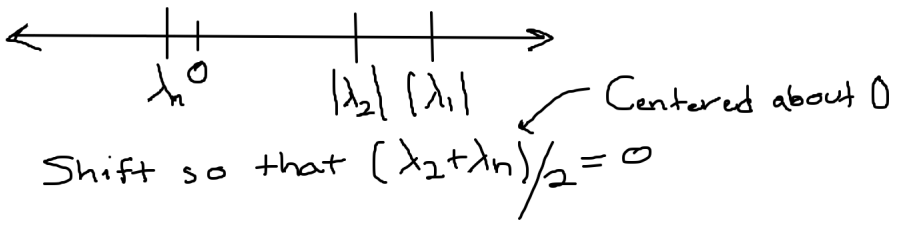
15.3 Shifted Power Method

Question: What are the eigenvalues of $A-\mu I$? We can use the perturbation of the Identity $(A-\mu I)u$ to show the following:

$$(A - \mu I)u = Au - \mu u = \lambda u - \mu u = (\lambda - \mu)u$$

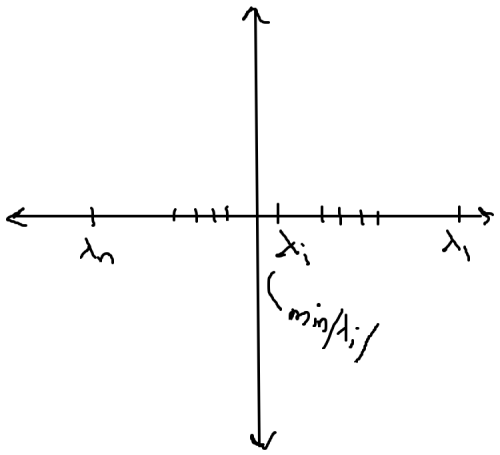
Conclusions:
If (λ_i, u_i) are A's eigenpairs, then $(\lambda_i - \mu, u_i)$ are $(A - \mu I)$'s eigenpairs.

Remember: For the power method, the convergence rate was controlled by the ratio $|\lambda_2/\lambda_1|$ assuming the eigenvalues are ordered like $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$



15.4 Inverse Power Method

Note: $Au = \lambda u \Rightarrow A^{-1}u = \frac{1}{\lambda}u$. So: if A has the eigenpairs (λ_i, u_i), then the eigenpairs of A^{-1} are $(1/\lambda_i, u_i)$.



Set the following:

$$\tilde{u}^{(k+1)} = A^{-1}\tilde{u}^{(k)}$$

$$u^{(k+1)} = \frac{\tilde{u}^{(k+1)}}{||\tilde{u}^{(k+1)}||}$$

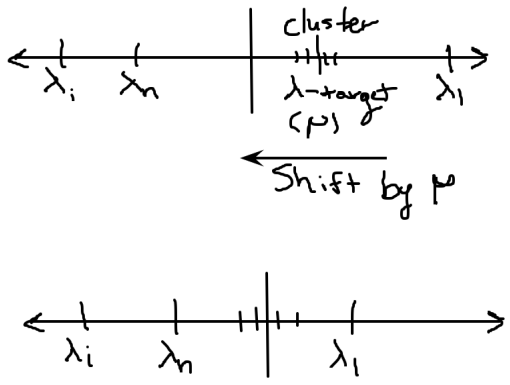
How does this work? Let $u^{(0)} = Qc = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = c_1u_1 + \cdots + c_nu_n$

Multiply: $A^{-1}u^{(0)} = (Q\Lambda Q^T)^{-1}u^{(0)}$

$= (Q^T)^{-1}\Lambda^{-1}Q^{-1}u^{(0)} = Q\Lambda^{-1}Q^Tu^{(0)}$ Note: Inverse of a diagonal matrix is diag$(\frac{1}{\lambda_1}, \cdots, \frac{1}{\lambda_n})$

Iterate:

$$\begin{aligned} A^{-1}u^{(0)} &= Q^{-1}\Lambda Q^T \cdots Q\Lambda^{-1}Q^T)u^{(0)} \qquad \text{(Iterate } k\text{-times)} \\ &= Q\Lambda^{-k}Q^Tu^{(0)} = Q\Lambda^{-k}Q^TQc = Q\Lambda^{-k}c \\ &= \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix} \begin{bmatrix} 1/{\lambda_1}^k & & \\ & \ddots & \\ & & 1/{\lambda_n}^k \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \\ &= \frac{c_1}{{\lambda_1}^k}u_1 + \cdots + \frac{c_n}{{\lambda_n}^k}u_n \end{aligned}$$



Exercise: Mimic the proof for the power method from the last lecture to show that as $k \to \infty$, only the u_n-term remains provided that $|\lambda_1| > \cdots > |\lambda_n|$

15.4.1 Computing the next eigenpair

Question: Once we've computed the maximum magnitude (λ_1, u_1)-pair, how do we compute (λ_2, u_2)?

Recall: $A = Q\Lambda Q^T = \sum_{i=1}^{n} \lambda_i u_i {u_i}^T$

Exercise: Show that the summation above is true.

Idea: Subtract $\lambda_1 u_1 {u_1}^T$ from A. So:

$$A - \lambda_1 u_1 {u_1}^T = \sum_{i=2}^{n} \lambda_i u_i {u_i}^T = \begin{bmatrix} u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} \lambda_2 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} u_2 & \cdots & u_n \end{bmatrix}^T$$

Note: This might be slow (deflation) and will not work for the inverse power method.

15.4.2 Additional Exercises on Eigenvalue Methods

1. Show that $\min_{u^T u = 1} u^T A u$ is equivalent to $Au = \lambda u$. (Hint: use Lagrange Multipliers)
2. See if you can show that $\min_{{u_1}^T u_1 = 1,\ {u_2}^T u_1 = 0} {u_2}^T A u_2$ is equivalent to finding the largest eigenvalue/eigenvector pair for $A - \lambda_1 u_1 {u_1}^T$

Chapter 16

Lagrange Interpolation

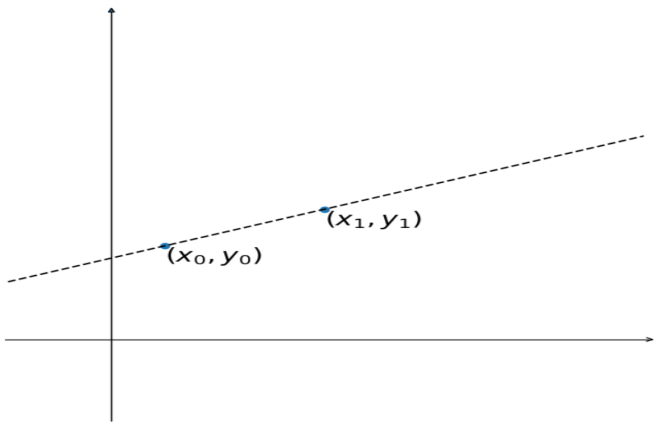
[SFP: Scribed by Yishi Wang]

16.1 Why Interpolation

1. Most functions aren't polynomial, like some special solution functions of ODEs and PDEs $(sin, cos, \sqrt{x}, exp, j_0, y_0)$.
2. But many of them are smooth, which we can efficiently interpolate them using polynomials.
3. Polynomials are things which can be put on a computer.

16.2 Polynomial Interpolation

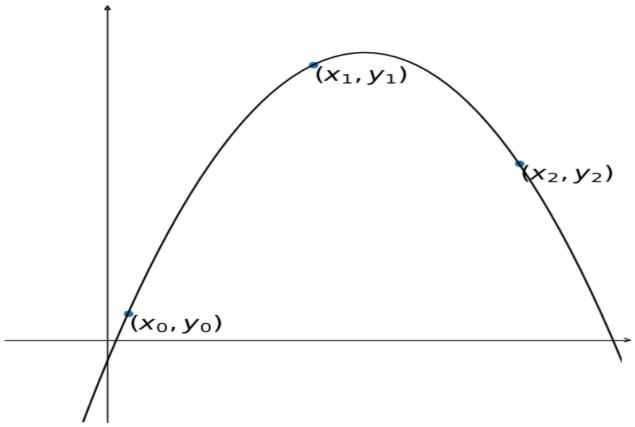
Sample: 2 points in $\mathbb{R}^2$



sample of degree one polynomial

- Two points in $\mathbb{R}^2$ forms a unique line
- A line is degree 1 polynomial: $p(x) = a_0 + a_1 x$

Sample: 3 points in $\mathbb{R}^3$



sample of degree two polynomial

- 3 points forms a unique quadratic(deg 2)
- $p(x) = a_0 + a_1 x + a_2 x^2$

Define: $\mathbb{P}^n = \{p(x) = a_0 + ax + a_2 x^2 + ... + a_n x^n \, : \, a_n \neq 0)\}$ = the degree of n polynomials.

Idea: $dim\ \mathbb{P}_n = n+1$ meaning that to fit (or interpolate) $p \in \mathbb{P}_n$ to some data, we need $n+1$ pieces of data.

Sample: fitting a cubic($p \in \mathbb{P}_n$) where $p(x) = a_0 + ax + a_2 x^2 + a_3 x^3$
let $x_0, x_1, x_2, x_3 \in \mathbb{R}$ [nodes]
let $y_0, y_1, y_2, y_3 \in \mathbb{R}$ [nodes]
Interpolate condition: $p(x_i) = y_i$ for i = 0, 1, 2, 3
write $y_i = a_0 + ax_1 + a_2 {x_2}^2 + a_3 {x_3}^3$ for i = 0, 1, 2, 3.
extend all values of i we get:
$$\begin{cases} y_0 = a_0 + a_1 x_0 + a_2 {x_0}^2 + a_3 {x_0}^3 \\ y_1 = a_0 + a_1 x_1 + a_2 {x_1}^2 + a_3 {x_1}^3 \\ y_2 = a_0 + a_1 x_2 + a_2 {x_2}^2 + a_3 {x_2}^3 \\ y_3 = a_0 + a_1 x_3 + a_2 {x_3}^2 + a_3 {x_3}^3 \end{cases}$$

Transform the above equations in to matrix form:

$$\begin{bmatrix} 1 & x_0 & {x_0}^2 & {x_0}^3 \\ 1 & x_1 & {x_1}^2 & {x_1}^3 \\ 1 & x_2 & {x_2}^2 & {x_2}^3 \\ 1 & x_3 & {x_3}^2 & {x_3}^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} (need \, : \, x_i \neq x_j \text{ , if } i \neq j) \tag{16.1}$$

Note:

$\begin{bmatrix} 1 & x_0 & {x_0}^2 & {x_0}^2 & {x_0}^3 \\ 1 & x_1 & {x_1}^2 & {x_1}^2 & {x_1}^3 \\ 1 & x_2 & {x_2}^2 & {x_2}^2 & {x_2}^3 \\ 1 & x_3 & {x_3}^2 & {x_3}^2 & {x_3}^3 \end{bmatrix}$ are called "Vandermonde Matrix".

16.2.1 Hermite Interpolation Problem

Sample: for cubic interpolation, if we have: $\begin{cases} x_0, y_0, y_0' \\ x_1, y_1, y_1' \end{cases}$, such that $\begin{cases} p(x_i) = y_i \text{ , i} = 0, 1 \\ p(x_i) = y_i \text{ , i} = 0, 1 \end{cases}$.

we first need to compute $p'(x)$:

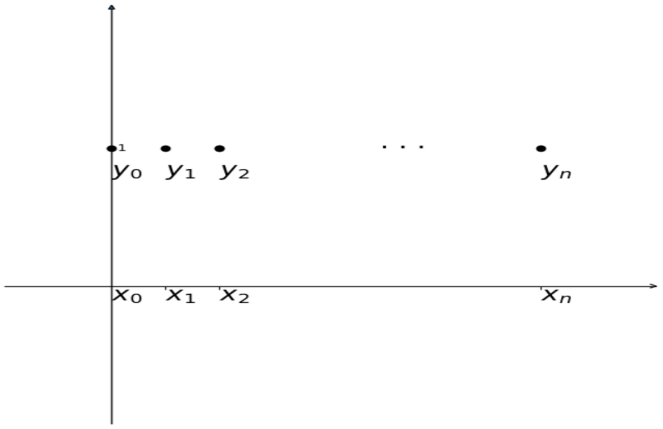
$p'(x) = (a_0 + ax + a_2x^2 + a_3x^3)' = a_1 + 2a_2x + 3a_3x^2$

The linear system then looks like below:

$$\begin{bmatrix} 1 & x_0 & {x_0}^2 & {x_0}^3 \\ 1 & x_1 & {x_1}^2 & {x_1}^3 \\ 0 & 1 & 2x_0 & 3{x_0}^2 \\ 0 & 1 & 2x_1 & 3{x_1}^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{16.2}$$

This is called a "Hermite interpolation problem".

Sample: transform the following Lagrange Interpolation Problem into matrix form.



Solution:

$$\begin{bmatrix} 1 & x_0 & {x_0}^2 & {x_0}^2 & ... & {x_0}^n \\ 1 & x_1 & {x_1}^2 & {x_1}^2 & ... & {x_1}^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ 1 & x_1 & {x_1}^2 & {x_1}^2 & ... & {x_1}^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{16.3}$$

Exercise: solve the cubic Hermite interpolation problem. **Hint:** First re-scale the problem so that $x_0 = 0$ and $x_1 = 1$, and unrescale after.

Note: we've seen function value interpolate conditions derivatives conditions, and we don't always get a "soluble" interpolation problem.

Sample: suppose we have function $p(x) = a_0 + a_1 x$ with conditions $\begin{cases} x_0, y_0' \\ x_1, y_1' \end{cases} \Rightarrow \begin{cases} p'(x_0) = y_0' \\ p'(x_1) = y_1' \end{cases}$,

which could be simplified to $\begin{cases} a_1 = y_0' \\ a_1 = y_1' \end{cases}$. There are two cases for solutions. The first solution is when $y_0' \neq y_1'$ implies the incompatible for initial condition. The second solution is when $y_0' = y_1$, which implies that $a_1 = y_0' = y_1'$. However, the value of a_0 could be any possible value for both cases.

Exercise: suppose we have function $p(x) = a_0 + a_1 x$ with conditions:

$$\begin{cases} x_0, y_0, y_0' \\ x_1, y_1, y_1' \end{cases} \quad (x_0 \neq x_1) \Rightarrow \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_0' \\ y_1' \end{bmatrix} \tag{16.4}$$

Solve this over determined linear system using one of the following techniques we learned:

1. QR Decomposition
2. normal equation(Cholesky decomposition)

Once you do this you have $\begin{cases} a_0 = a_1(x_0, x_1, y_0, y_1, y_0', y_1') \\ a_1 = a_1(x_0, x_1, y_0, y_1, y_0', y_1') \end{cases}$.

Vandermonde is often poorly condition, so we usually don't want to invert, but explore other option to solve interpolation polynomials.

Goal: come up with the Lagrange form of the interpolation polynomial. This is a specific type of interpolation problem with "pure interpolation constrains" (function values only):

$$\begin{cases} p(x_i) = y_i \text{ for } i = 1, ..., n \ (p \in \mathbb{P}^2) \\ \text{assuming } i \neq j, x_i \neq x_j \end{cases}$$

Idea: construct a special basis for $\mathbb{P}_n$. What we want are polynomials $L_i \in \mathbb{P}_n$, such that $L_i = \delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ else} \end{cases}$. Then, we have Lagrange form of the interpolation polynomial $p(x_i) = \sum_{i=1}^{n} y_i L_i(x)$.

Observe: $p(x_i) = \sum_{i=1}^{n} y_i L_i(x_i) = \sum_{i=1}^{n} y_i \delta_{ij} = y_i$

Exercise: let $w(x) = (x - x_0)(x - x_1) \dots (x - x_n) = \prod_{i=0}^{n}(x - x_i)$, show that $L_i = \frac{w(x)}{(x-x_i)w'x_i}$.

Exercise: show that $L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$ and verify that $L_i(x_j) = \delta_{ij}$.

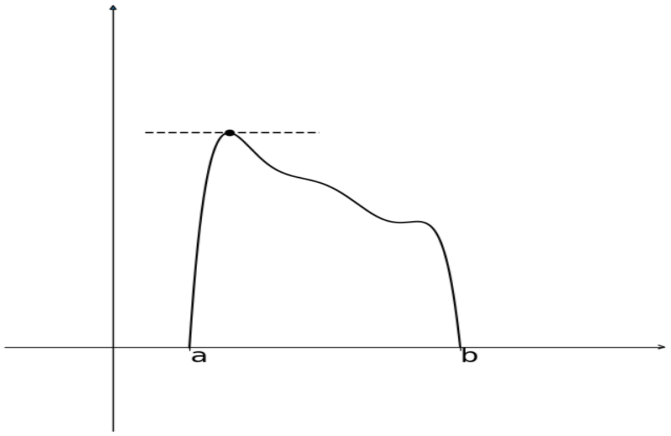
Exercise: show that p is unique.

16.2.2 Rolle's Theorem

Theorem: Let f being interpolated, and being in $\mathbb{C}^(n+1)[a, b]$, where $a = x_0 < x_1 < \dots < x_n = b$. Then, for every $x \in [a, b]$, where exists $\xi \in (a, b)$, where $\xi = \xi(x)$, such that $f(x) - p(x) = \frac{f^{n+1}\xi}{(n+1)!} w(x)$.

Proof: If $x = x_i$ for some i, then $f(x_i) - p(x_i) = 0$. Let $g(t) = p(t) = f(t) - \frac{w(t)}{w(x)}(p(x) - f(x))$. We can observe that g has n^{th} zeros at $x_0, x_1, \dots x_n$.

If $f \in \mathbb{C}'[a, b]$, and $f(a) = 0 = f(b)$, then there exist $x \in [a, b]$ such that $f'(x) = 0$.



example of Rolle's Theorem

Plan of attack: compute $g^{n+1}(t)$, where $g(t) = p(t) - f(t) - \frac{w(t)}{w(x)}(p(x) - f(x)) \Rightarrow g^{n+1} = p^{n+1}(t) - f^{n+1}(t) - \frac{w^{n+1}(t)}{w(x)}(p(x) - f(x))$.

Exercise: proof $p^{n+1}(t) = 0$ (Hint: $deg\ p = n$)

Exercise: Since $w(t) = (t - x_0)(t - x_1) \dots (t - x_n) \Rightarrow w'(t) = \sum_{j=0}^{n} \prod_{i=0\ i\neq j}^{n}(t - x_i)$, if we doing this recursively we could get $w^{n+1}(t) = (n + 1)!$

Chapter 17

Lagrange interpolation (cont.)

[SFP: Scribed by Yilin Li 04/06]

17.1 Note on PHW3

Do not use np.linalg.inv (because it takes more memory and time, also it is inaccurate)
1. Use scipy.sparse.csr.matrix csc dok
2. Use scipy.sparse.linalg.splu to compute a sparse LU decomposition

17.2 Lagrange Interpolation(continue.)

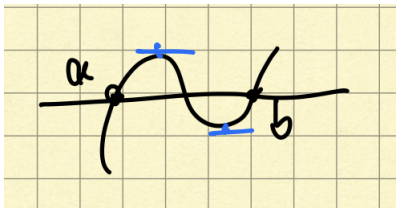
Theorem
Let f the function being interpolated to be in $C^{n+1}[a,b]$ where $a = x_0 < x_1 < ... < x_n = b$. Then, for every $x \in [a,b]$, there exists $\xi \in (a,b)$ s.t.

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w(x) \tag{17.1}$$
$$\text{where } \; w(x) = (x - x_0)(x - x_1)...(x - x_n)$$

Proof
If $x = x_i$ for some i, then $f(x_i) - p(x_i) = 0$. Define function $g(t,x)$ where x is fixed

$$g(t) = p(t) - f(t) - \frac{w(t)}{w(x)}(p(x) - f(x)) \tag{17.2}$$

Rolle's Theorem If $f \in C^2[a,b]$ and $f(a) = 0 = f(b)$, then there exists $\xi \in [a,b]$ s.t. $f'(\xi) = 0$

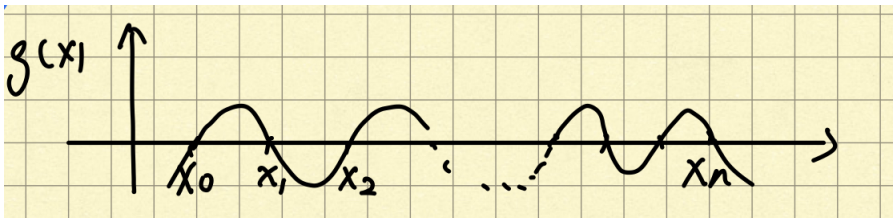


compute $g^{(n+1)}$

$$\begin{aligned} g^{(n+1)} &= p^{(n+1)}(t) - f^{(n+1)}(t) - \frac{w^{(n+1)(t)}}{w(x)}(p(x) - f(x)) \\ &= -f^{(n+1)}(t) - \frac{(n+1)!}{w(x)}(p(x) - f(x)) \end{aligned} \tag{17.3}$$

Note

1. $p^{(n+1)}(t) = 0$ because $deg\ p = n$
2. $w^{(n+1)}(t) = (n+1)!$ Prove **Exercise 1**



Observation

- g has (n+2) zeros
- g' has (n+1) zeros
- ...
- $g^{(n+1)}$ has 1 zero

So, $\exists \xi \in (a, b)\ :\ g^{(n+1)}(\xi) = 0$. Plug in equation(3), we have

$$0 = g^{(n+1)}(\xi) = -f^{(n+1)}(t) - \frac{(n+1)!}{w(x)}(p(x) - f(x)) \tag{17.4}$$

Then, we will get equation (1). Done.

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}w(x)$$

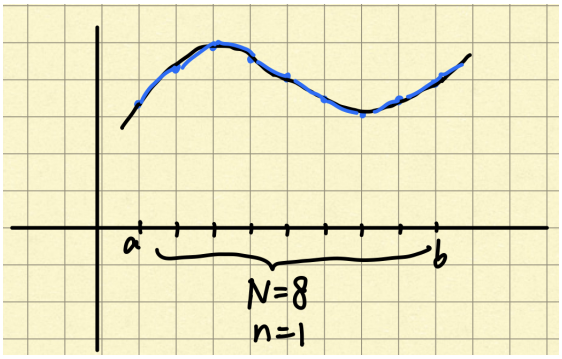
What does this do for us?

1. Assume $|x_n - x_0| = h$, then $|w(x)| = |(x - x_0)(x - x_1)...(x - x_n)| \leq h^{n+1}$.
2. $f^{(n+1)}$ also controls the error

17.3 Piece-wise Polynomial Interpolation

Runge Phenomenon
There are counterexamples where, for uniformly spaced x_i's, the error does not go to zero as $n \to \infty$
Observation: the error is sensitive to the node distribution and the best node distribution depends on f.

Piece-wise polynomial interpolation
—-a way to avoid Runge Phenomenon, it can be any interpolation, here we take the Lagrange interpolation as an example
Idea: chop $[a, b]$ up into N sub-intervals, apply Lagrange interpolation over each sub-interval with (n+1) nodes.



Eg. piecewise linear interpolation

$$p(x) = \begin{cases} p_1(x) & x_0 \le x \le x_1 \\ p_2(x) & x_1 \le x \le x_2 \\ \dots & \\ p_N(x) & x_{N-1} \le x \le x_N \end{cases}$$

$$p_i(x) = f(x_{i-1})L_0^i(x) + f(x_i)L_1^i(x) \tag{17.5}$$

As proved in previous exercise

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{17.6}$$

$$\begin{cases} L_0^i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_{i+j}}{x_{i-1+j} - x_{i+j}} = \frac{x - x_i}{x_{i-1} - x_i} \\ L_1^i(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}} \end{cases} \tag{17.7}$$

So,

$$\begin{cases} L_0^i(x_{i-1}) = \frac{x_{i-1}-x_i}{x_{i-1}-x_i} = 1 \\ L_0^i(x_i) = \frac{x_i-x_i}{x_{i-1}-x_i} = 0 \\ L_1^i(x_{i-1}) = \frac{x_{i-1}-x_{i-1}}{x_i-x_{i-1}} = 0 \\ L_1^i(x_i) = \frac{x_i-x_{i-1}}{x_i-x_{i-1}} = 1 \end{cases} \tag{17.8}$$

Exercise 2 write down piece-wise LI for n=2 and plot the L_j's by hand.

Assume that $|x_i - x_{i-1}| = h > 0$, apply equation(1) to each sub-interval: For $x_{i-1} \leq x \leq x_i$:

$$f(x) - p_i(x) = \frac{f''(\xi_i)}{2}(x - x_{i-1})(x - x_i) \qquad \text{where } x_{i-1} < \xi_i < x_i \tag{17.9}$$

Note: $f''(\xi_i)$ is hard to find
So, we **define notation**:

$$\|f - p\|_{[x_{i-1},x_i]} = max_{x_{i-1}\leq x\leq x_i}|f(x) - p_i(x)| \tag{17.10}$$

Goal: find $\|f - p\|_{[a,b]}$ **Start with** $\|f - p\|_{[x_{i-1},x_i]}$

$$\begin{aligned} |f(x) - p_i(x)| &= \frac{|f''(\xi_i)|}{2}|x - x_{i-1}||x - x_i| \\ &\leq \frac{|f''(\xi_i)|}{4}h^2 \\ &\leq \frac{\|f''\|_{[x_{i-1},x_i]}}{4}h^2 \end{aligned} \tag{17.11}$$

Exercise 3 Show $|x - x_{i-1}||x - x_i| \leq \frac{h^2}{2}$
Exercise 4 Show $\|f - p\|_{[a,b]} \leq \frac{\|f''\|_{[a,b]}h^2}{4}$
Exercise 5 What happens for this example when f is not in $C^2[a, b]$?
Exercise 6 Approximate the integral: $I = \int_a^b f(x)dx$ with the integral: $I_N = \int_a^b p(x)dx$. Then

1. Compute the error $E_N = I_N - I$
2. Show that this is just the Trapezoid Rule

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate
import scipy.special

# function we want to interpolate
f = lambda x: 1/(1 + 5*x**2)

N = np.arange(1, 21)
E = np.empty(N.size, dtype=np.float64)

n = 9

x = np.linspace(-1, 1, n + 1) # Find a set of equally spaced nodes
xr = np.sort(np.random.uniform(-1, 1, n + 1)) # Find a set of random nodes

# Return a Lagrange interpolating for the nodes chosen
p = scipy.interpolate.lagrange(x, f(x))
pr = scipy.interpolate.lagrange(xr, f(xr))

# test on an special function: the value of the first node is 1 and 0 for the other
e = np.zeros_like(x)
e[0] = 1
L3 = scipy.interpolate.lagrange(x, e)

# plot the interpolation result p(x) compare to the function f(x)
ngrid = 501
xgrid = np.linspace(-1, 1, ngrid)
fgrid = f(xgrid)
pgrid = p(xgrid)
prgrid = pr(xgrid)
L3grid = L3(xgrid)

# Runge's phenomenon can be observed at the edge of the interval,
# where increasing n does not improve accuracy.
# Error does not converge to zero when n approaches to infinity
plt.figure()
plt.plot(xgrid, fgrid, label=r'$f(x)$', zorder=1, linewidth=2)
plt.plot(xgrid, pgrid, label=r'$p(x)$', zorder=2, linestyle='--',
         linewidth=2)
plt.plot(xgrid, prgrid, label=r'$p(x)$ (random)', zorder=2, linestyle='--',
         linewidth=2)
```

```
# plt.plot(xgrid, L3grid)
plt.scatter(x, e, s=20)
plt.legend()
plt.show()
```

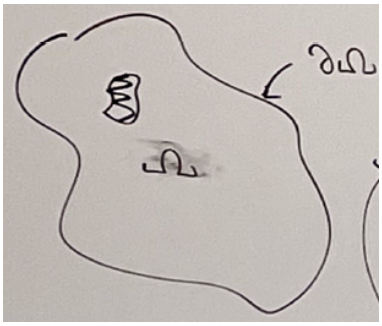
Chapter 18

Function Approximation

[SFP: Scribed by Jiahao Hui 4/11]

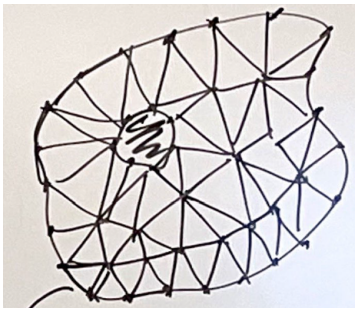
Hermite Interpolation (and Spines)
"Piecewise Lagrange"... we don't hear about this so often because usually you see it in the Finite element method (FEM).

18.1 Quick intro to FEM:

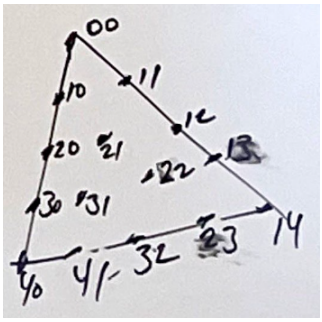


Poisson's equation *w/* Dirichlet BC's $\begin{cases} \Delta u = f \text{ on } \Omega \\ u = y \text{ on } \partial\Omega \end{cases}$

1) Build mesh on Ω :

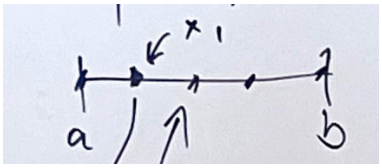


$\longrightarrow$



$\longleftarrow \quad Lij(x, y)$

Compare $\omega/1D$:



$$p \in \mathbb{P}_4$$

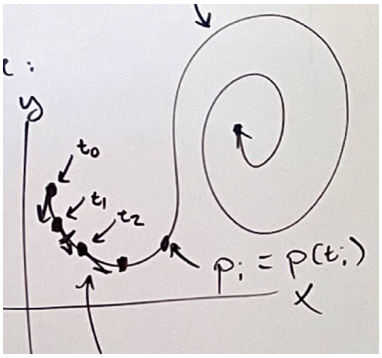
$$L_1(x) = \begin{cases} 1 & \text{if } x = x_1 \\ 0 & \text{else} \end{cases}$$

2) Connect Poisson's equation into variational form to solve

Splines:

Example:

$$p(t) = (x(t), y(t))$$



$$p_i' = p_i'(t)$$
$$t_i = \frac{p_i'(t)}{\left\|p_i'(t)\right\|}$$

If we have:

$$t_0 < t_1 < \cdots < t_N, \quad N > 0$$
$$p_i, \quad i = 0, \ldots, N$$
$$p_i', \quad i = 0, \ldots, N$$

then we can solve the Piecewise Hermite interpolation problem:

$$\hat{p}(t) = \begin{cases} \hat{p}_1(t) \text{ if } t_0 \le t \le t_1, \\ \vdots \\ \hat{p}_N(t) \text{ if } t_{N-1} \le t \le t_N \end{cases}$$

Where:
$\hat{p}_i(t) = p_{i-1}L_0(\frac{t-t_i}{\Delta t_i}) + p_iL_1(\frac{t-t_{i-1}}{\Delta t_i}) + p_{i-1}'\Delta t_iH_0t_i(\frac{t-t_{i-1}}{\Delta t_i}) + p_i'\Delta t_iH_1(\frac{t-t_{i-1}}{\Delta t_i})$

Note: if $t_{i-1} \le t \le t_i$, then $\frac{t-t_{i-1}}{\Delta t_i} \in [0, 1]$.

The L_i's and H_i's satisfy:

$$\begin{array}{c|c|c|c} L_0(0) = 1 & L_1(0) = 0 & H_0(0) = 0 & H_1(0) = 0 \\ L_0(1) = 0 & L_1(1) = 1 & H_0(1) = 0 & H_1(1) = 0 \\ L_0'(0) = 0 & L_1'(0) = 0 & H_0'(0) = 1 & H_1'(0) = 0 \\ L_0'(1) = 0 & L_1'(1) = 0 & H_0'(1) = 0 & H_1'(1) = 1 \end{array}$$

18.2 Function approximation

Quick note: We saw that uniformly spaced nodes for Lagrange interpolation applied to $f(x) = \left(1 + 5x^2\right)^{-1}$ did not converge as the degree of the L interpolation. $\to \infty$.

Why? $f(x) = \left(1 + a^2x^2\right)^{-1}$ has a singularity at $x = a^{-1}i$, $\left(\left(a^{-1}i\right)^2 = a^{-2}i^2 = -a^{-2}\right)$ Thi. means that the Taylor series approximation f has its radius of convergence limited by the position of this singularity.

Eg. $\dfrac{1}{1 + a^2x^2} = 1 - a^2x^2 + a^4x^4 - a^6x^6 + \cdots$

18.3 Norms on some function Spaces:

$$f \in C^0[a, b], \quad a < b$$

Define:

$$\|f\|_\infty = \max_{a \leqslant x \leqslant b} |f(x)|$$

$$\|f\|_2 = \left[\int_a^b |f(x)|^2 dx\right]^{1/2}$$

$$\|f\|_1 = \int_a^b |f(x)| dx$$

These setisfy the some properties as the Vector norms that we saw already:

1) $\|f\| = 0 \iff f \equiv 0$

2) $\alpha \in \mathbb{R}$; $\|\alpha f\| = |\alpha| \cdot \|f\|$

3) $\|f + g\| \leq \|f\| + \|g\|$

can generalise a bit. Let $w \in C^0[a, b]$ s.t. $w(x) > 0$ if $a < x < b$ be a weight function, then we can define e.g. a weighted version of these norms:

$$\|f\|_{2,w} = \left[\int_a^b f(x)^2 w(x) dx\right]^{1/2}.$$

Similar idea in linear algebra.

Exercise:
Let A be an spd matrix. Show that:

$$\|x\|_A = \sqrt{x^t A x}$$

is a norm on $\mathbb{R}^n$ (or whatever). This is called a weighted vector norm. Explain how this relates to "$\|f\|_{2,w}$ ".

Note:

$$\|f\|_{2,w}^2 = (f, wf) = \int_a^b f(x)w(x)f(x)dx$$
$$= \int_a^b f(x)^2 w(x)dx.$$

The distance between two functions:

$$\|f - g\|$$

Note:
if $\|f - g\| = 0$, then $f = g$.

The goal in function approximation is:

1) let p be finite dimensional,

   e.g. $p \in \mathbb{P}_n$

1.5) pick $[a, b]$

2) try to minimize $\|f - p\|$

3) pick w

Solve:
$\min_{p\in\mathbb{P}_n} \|f - p\|_\infty$

From analysis, there is the Weiestrass approximation theorem, which says for any $f \in C^\circ(a, b)$, and any $\varepsilon > 0$, there exists (at least one!) polynomial p such that $\|f - p\|_\infty < \varepsilon$.

Problem: No control over degree of P. Could be huge.

Instead: Fix n. See what we can do.

$$= \underbrace{\begin{matrix}\min & \max \\ p \in \mathbb{P}_n & a \le x \le b\end{matrix}}_{\text{=minimax polynomial interpolation}} |f(x) - p(x)|$$

Note:
If $f \in C^\circ[a, b]$, then there is a unique solution which obtains the minimum value. But it depends on f and it is basically impossible to get.

Different approach: solve the minimax problem for $f = x^{n+1}$.

$$\begin{matrix}\min & \max \\ p \in \mathbb{P}_n & a \le x \le b\end{matrix} |x^{n+1} - p(x)|$$

Theorem:
let $n > 0$, let $[a, b] = [-1, 1]$, then $p(x) = x^{n+1} - \frac{1}{2^n} T_{n+1}(x)$ is the solution of this problem, where:

$$\underbrace{T_n(x) = \cos(n \cos^{-1} x).}_{\text{This is called the degree } n \text{ Chebyshev poly.}}$$

Note:
$\frac{1}{2^n} T_{n+1}(x)$ is a monic polynomial (leading coefficients=1), and $\left\| \frac{1}{2^n} T_{n+1}(x) \right\|_{[-1,1]}$ is minimal over all degree $n + 1$ monic polynomial.

Property:

$$\begin{aligned} T_0(x) &= 1, \quad (\text{ for } [a, b] = [-1, 1]) \\ T_1(x) &= x \\ &\vdots \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x). \end{aligned}$$

Eg:

$$\begin{aligned} T_2(x) &= 2xT_1(x) - T_0(x) \\ &= 2x \cdot x - 1 = 2x^2 - 1 \end{aligned}$$

Exercise:
T_n is deg n. So, should have n zeros … show that the zeros of T_n are: $x_j = \cos\left(\frac{(2j-1)\pi}{2n}\right)$ for $j = 1, \ldots, n$.

Claim:
if we use the zeros of the T_n's as Lagrange interpolation nodes (over $[-1, 1]$) then, the error is close to optimal.

Idea:
Approx error for $x \in [-1, 1]$, $\exists\, \xi \in (-1, 1)$ s.t.

$$f(x) - \underbrace{p(x)}_{\substack{\uparrow \\ \text{Lagrange interpolation} \\ w/.\ T_n\text{'s zeros as} \\ \text{interp. nodes}}} = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=1}^{n} (x - x_j)$$

$$\uparrow \qquad\qquad \uparrow$$

$$= \cos\left(\frac{(2j-1)\pi}{2n}\right)$$

But:

$\prod_{j=1}^{n}\left(x-x_j\right)=\frac{1}{2^{n-1}}T_n(x)$
(Why? If $p\in\mathbb{P}_n$, and its roots at x_j, then $\prod_j\left(x-x_j\right)=p(x)$)

Recap:

1) We want to do Lagrange Interpolation over $[-1,1]$
2) We know the errors is very sensitive to the choice of nodes
3) We have an exact expression for the point-wise error

Exercise: let $w(x)=\frac{1}{\sqrt{1-x^2}}$ be a weight function on $[-1,1]$. Show that:

$$\begin{aligned}\left(T_m,T_n\right)_\omega &= \int_{-1}^{1} T_m(x)T_n(x)\omega(x)dx\\ &= \begin{cases}0 & \text{if } m\neq n\\ s\pi & \text{if } m=n=0\\ \frac{\pi}{2} & \text{if } m=n\neq 0\end{cases}\end{aligned}$$

$\Rightarrow$ the Chebyshev polys are a "system of orthogonal polyromials" on $[-1,1]$, *where* $\omega(x)=\frac{1}{\sqrt{1-x^2}}$.

Chapter 19

More Vandermonde

[SFP: Scribed by Danny Chen, April 20nth 2022]

Reminder:

Let $\{x_i\}_{i=0}^n \subseteq \mathbb{R}$, then the Vandermonde matrix(for Lagrange interpolation) is:

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix}$$

Prop:

$det(V) = 0 \Leftrightarrow x_i \neq x_j$ if $i \neq j$ $det(V) = 0 \Leftrightarrow$ there exists a pair i, j st $i \neq j$ and $x_i = x_j$

The correct part of my wrong proof:

Assume $\exists i \neq j$, $x_i = xj$. Write:

$$V = \begin{bmatrix} r_0 \\ r_1 \\ ... \\ r_n \end{bmatrix}, r_i = \begin{bmatrix} 1 & x_i & x_i^2 & ... & x_i^n \end{bmatrix}$$

Look at r_i and r_j. Clearly, since $x_i = x_j$,$r_i = r_j$. So, two rows at V are equal, hence, there is a subset of rows of V which is linearly dependent, hence $det(v) = 0$.

The wrong part of My wrong proof:

If $det(V) = 0$, then exists a subset st linearly dependent rows of V. That means, exists an index set $I \subseteq \{0, 1, ..., n\}$ and a vector of linear coefficients $\alpha = (\alpha_i)_{i \in I} \in \mathbb{R}^{|I|}$. such that $\alpha \neq 0$ and $\sum_{i \in I} \alpha_i r_i = 0$, we could also write this like $V_I^T \alpha = 0$, where $V_I = [r_i]_{i \in I} \in \mathbb{R}^{|I| \times (n+1)}$.(So far so good)

Let's write $\sum_{i \in I} \alpha_i r_i = 0(\mathbb{R}^{n+1})$ explicitly:

$$\begin{cases} \sum_{i \in I} \alpha_i \cdot 1 = 0(1^{st}\ component) \\ \sum_{i \in I} \alpha_i \cdot x_i = 0(2^{nd}\ component) \\ \sum_{i \in I} \alpha_i \cdot x_i^2 = 0(3^{rd}\ component) \\ ... \\ \sum_{i \in I} \alpha_i \cdot x_i^n = 0(n+1^{st}\ component) \end{cases}$$

Up until here the proof is still okay. Simple example in x, y, z (instead x_i) and α, β, γ (instead α)

$$\begin{cases} \alpha + \beta + \gamma = 0 \\ \alpha x + \beta y + \gamma z = 0 \\ \alpha x^2 + \beta y^2 + \gamma z^2 = 0 \end{cases}$$

Exercise
explain why we have made no progress with this proof.(hint. what are we trying to proof?)

Nigle's correct proof:

$$VC = \begin{bmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & ... & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ ... \\ c_n \end{bmatrix}$$

$\Rightarrow$ same as before.

$\Rightarrow$ Assume $det(V) = 0$. A consequence of this is that there exist a vector of polynomial coefficients $c \in \mathbb{R}^{n+1}$ such that $V \cdot C = 0 \in \mathbb{R}^{n+1}$.

Define:
$p(x) = c_0 + c_1 x + ... + c_n x^n$

Notice that the i^{th} entry of VC looks like:

$$0 = (VC)_i = c_0 + c_1 x_i + ... + c_n x_i^n = p(x_i)$$

So we have

$$0 = VC = \begin{bmatrix} p(x_0) \\ p(x_1) \\ ... \\ p(x_n) \end{bmatrix}$$

Hence, $p(x_i) = 0 \; for \; i = 0, ..., n$.

On the other head, $p \in \mathbb{P}_n (degp = n)$. By the fundamental theorem of algerbra, p has at most n real roots. But evidently, all x_i are roots of P. We conclude that the set of interpolation nodes $x_{i\,i=0}^{\;n}$ must have cardinality at most n. The only thing to conclude is that there must be at least a pair of nodes $x_i, x_j (i \neq j)$ which are equal.

Te be clear:

1) multiplying V by a coefficient vector C, evaluates the polynomial corresponding to C at each of the interpolation nodes defining V.

2) if $f = V_c$, and f is known but C is not, then $C = V^{-1} f$ solves the interpolation problem.

3) Previously, we saw the Lagrange basis functions: $L_i \in \mathbb{P}_n$ s.t. $L_i(x_j) = \delta_{i,j}.(Li_x = \prod_{i=0, j\neq 1}^{n} \frac{x - x_j}{x_i - xj})$ what this mean is that to solve the interpolation problems: find $p \in \mathbb{R}_n$ such that $p(x_i) = f(x_i)$. we can just write:

$$p(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + ... + f(x_n)L_n(x)$$

A basis(u_1) is a set of element which spans a vector space.

$\forall x \; \in \$thevectorspace, there \exists$ a vector of cofficients α_i s.t $x = \sum \alpha_i u_i$

(Note: sometimes the word “frame” is used if the basis has redindant elements)

So:
$\{L_i\}_{i=0}^n$ span $\mathbb{P}_n$
$\Rightarrow$ for any $p \in \mathbb{P}_n$, $\exists\{\alpha_i\}$ st $p = \alpha_0 L_0 + \alpha_1 L_1 + ... + \alpha_n L_n$
Another basis is the "monomial basis":$\{1, x, x^2, ..., x^n\}$
(this is the "standard" basis for a polymonial in $\mathbb{P}_n$)

We can think of the Vandermonde matrix as a transform from polynomial coefficients to function values(evaluated polynomials).

$c \mapsto f = V_c$ (Forward transform)

$f \mapsto c = V^{-1} f$ (Inverse transform)

Let's look at L_i, we know "f_i" for L_i, but we don't know c.

If c is the vector of "coefficients" for the polynomial $p \in \mathbb{P}_n$, then:

$$f = (p(x_0), p(x_1), p(x_2), ..., p(x_n))$$

So, "f_i" for L_i is:

$$f_i = (0, 0, ..., 0, 1, 0, ..., 0) = e_i$$

(The value "1" is on the i^{th} position, e_i is "standard basis vector for $\mathbb{R}^{n+1}$")

Then, "c" for L_i is:

$$c_i = V^{-1} e_i$$

(Where i is the i^{th} column of V^{-1})

why?

$$\begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ ... & ... & ... \\ a_{n1} & ... & a_{nn} \end{bmatrix} \begin{bmatrix} 0 \\ ... \\ 1 \\ ... \end{bmatrix} \Leftarrow i^{th}\ place = \begin{bmatrix} a_{11} \\ ... \\ ... \\ a_{n1} \end{bmatrix}$$

This tells me that the columns of V^{-1} are the "coefficients" of $L_0, ..., L_n$.

$$V^{-1} = [c_0 \quad c_1 \quad ... \quad c_n]$$

So it's not enough to talk about "coefficients" of a vector, you must specify the basis. Coefficients are always with respect to a basis.

Exercise:
What are the coefficient of L_i in the Langrange basis?($\{L_0, ..., L_n\}$)

Exercise:
What are the coefficient of L_i in the monomial basis?($\{1, x, ..., x^n\}$)

1. Prove that $D^4u = f$, st $u(a) = u(b) = u'(a) = u'(b) = 0$ has unique solution by first proving that

$$\int_a^b u(t)D^4u(t)dt = 0, \ if \ and \ only \ if \ u \equiv 0$$

(4 means integrate by parts 4 times.)
And then use this to prove the existence and uniqueness of the cubic Hermit polynomial $p(t)$.

3. Quintic Hermite polys exists ... and solve:

$$p^{(k)}(a) = f^{(k)}(a)$$
$$p^{(k)}(b) = f^{(k)}(b)$$
$$for \ k = 0, 1, 2$$

Chapter 20

Hermite Interpolation

[SFP: Scribed by Richen Du.]

20.1 More interpolation: Hermite mainly...

Let's look at Hermite inerpolation again.
In general, you can do p^{th} order Hermite interpolation on the interval $[x_0, x_1]$:
find $p \in \mathbb{P}_{2p+1}$ such that $p^{(k)}(x_i) = f^{(k)}(x_i)$, for $i = 0, 1$, for $k = 0, ..., p$
(*Goal: find p which interpolates function values and derivatives up to some order p at x_0 and x_1)

Example: $p = 2$. Find $p \in \mathbb{P}_{2*2+1} = \mathbb{P}_5$ such that
$p(x_0) = f(x_0)$, $p(x_1) = f(x_1)$
$p'(x_0) = f'(x_0)$, $p'(x_1) = f'(x_1)$
$p''(x_0) = f''(x_0)$, $p''(x_1) = f''(x_1)$
this is called a "quintic Hermite interpolation"

The Vandermonde matrix for this problem (so: for $p = 2$, and x_0, x_1 fixed):

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & x_0^4 & x_0^5 \\ 1 & x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ 0 & 1 & 2x_0 & 3x_0^2 & 4x_0^3 & 5x_0^4 \\ 0 & 1 & 2x_1 & 3x_1^2 & 4x_1^3 & 5x_1^4 \\ 0 & 0 & 2 & 6x_0 & 12x_0^2 & 20x_0^3 \\ 0 & 0 & 2 & 6x_1 & 12x_1^2 & 20x_1^3 \end{bmatrix} \tag{20.1}$$

What's special about V?
If $\alpha = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ is the vector of polynomial coefficients for $p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 +$

$\alpha_3 x^3 + \alpha_4 x^4 + \alpha_5 x^5$, then

$$V_\alpha = \begin{bmatrix} p(x_0) \\ p(x_1) \\ p'(x_0) \\ p'(x_1) \\ p''(x_0) \\ p''(x_1) \end{bmatrix} \tag{20.2}$$

"Multiplication by V" $\leftrightarrow$ "polynomial evaluation"

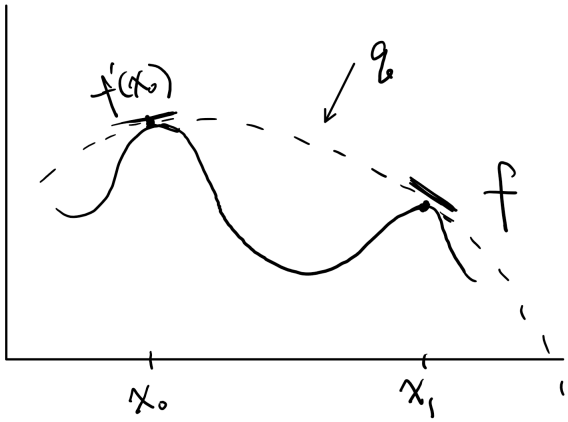
OTOH: if $f = (f(x_0), f(x_1), f'(x_0), f'(x_1), f''(x_0), f''(x_1))$, (where f is just some e^2 function) then I can write: "$\beta = V^{-1} f$"
Let's say we now let β define a polynomial:

$$q(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4 + \beta_5 x^5 \tag{20.3}$$

Then: what is V_β?

$$\begin{bmatrix} f(x_0) \\ f(x_1) \\ f'(x_0) \\ f'(x_1) \\ f''(x_0) \\ f''(x_1) \end{bmatrix} = f = VV^{-1} f = V_\beta = \begin{bmatrix} q(x_0) \\ q(x_1) \\ q'(x_0) \\ q'(x_1) \\ q''(x_0) \\ q''(x_1) \end{bmatrix} \tag{20.4}$$

Started with:



20.2 The cardinal basis (for an interpolation problem)

Example: $p = 2$, quintic Hermite interpolation:
Can I find polys like the Lagrange basis functions for this problem?

(*monomial basis for $\mathbb{P}_5$: $\{1, x, x^2, x^3, x^4, x^5\}$)
(*"cardinal basis": $\{H_{00}, H_{01}, H_{10}, H_{11}, H_{20}, H_{21}\}$)

$p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \alpha_4 x^4 + \alpha_5 x^5$
$= f(x_0)H_{00}(x) + f(x_1)H_{01}(x) + f'(x_0)H_{10}(x) + f'(x_1)H_{11}(x) + f''(x_0)H_{20}(x) + f''(x_1)H_{21}(x)$
such that

$$H_{ij}^{(l)}(x_k) = \delta_{jk}\delta_{il} \tag{20.5}$$

for $i = 0, 1, 2$; $j = 0, 1$; $k = 0, 1$; $l = 0, 1, 2$.
(*i is the order of the derivative, and j is the endpoint)

What's the point?
→ in the cardinal basis, the coefficients of the polynomial p which solves the interpolation problem are just "the data": the values $(f(x_0), f(x_1), f'(x_0), f'(x_1), f''(x_0), f''(x_1))$

Why do we care?

- using the Vandermonde matrix can be ineffective. If we change x_0 or x_1, have to redo everything...
- OTOH: $H_{ij}(x) = H_{ij}(x; x_0, x_1)$
- it's clear how the data affect the polynomial p...if I change one piece of data, only affects one coefficient.

How do we compute them?

- find coefficients of each H_{ij} in the monomial basis, i.e., find α_k^{ij} such that
  $H_{ij}(x) = \alpha_0^{ij} + \alpha_1^{ij} x + \alpha_2^{ij} x^2 + \alpha_3^{ij} x^3 + \alpha_4^{ij} x^4 + \alpha_5^{ij} x^5$
- use the Vandermonde matrix:
  $$\begin{bmatrix} H_{ij}(x_0) \\ H_{ij}(x_1) \\ H'_{ij}(x_0) \\ H'_{ij}(x_1) \\ H''_{ij}(x_0) \\ H''_{ij}(x_1) \end{bmatrix} = f^{ij} = V\alpha^{ij}$$

Exercise: for $p = 2$:

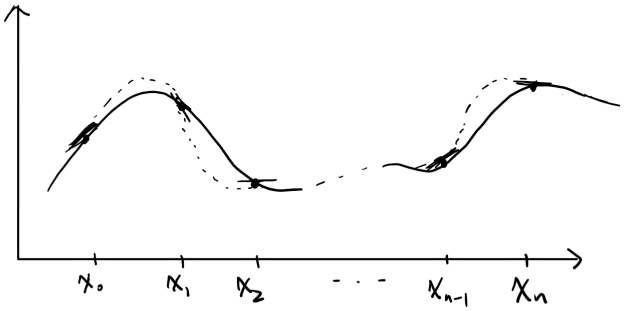
$$V \begin{bmatrix} \alpha^{00} & \alpha^{01} & \alpha^{10} & \alpha^{11} & \alpha^{20} & \alpha^{21} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{20.6}$$

Or: $VA = I \Rightarrow A = V^{-1}$
So: $\begin{bmatrix}\alpha^{00} & \alpha^{01} & \alpha^{10} & \alpha^{11} & \alpha^{20} & \alpha^{21}\end{bmatrix} = V^{-1}$

(*In other words: the columns of V^{-1} are the coefficients of the cardinal basis functions in the monomial basis.)

20.3 Piecewise Hermite interpolation

Piecewise Hermite interpolation is simple:
just do Hermite interpolation over each subinterval.



For a subinterval $[x_{i-1}, x_i]$, define:

$$\begin{aligned} p_i(x) = {} & f(x_{i-1})H_{i00}(x) + f(x_i)H_{i01}(x) \\ & + f'(x_{i-1})H_{i10}(x) + f'(x_i)H_{i11}(x) \\ & + \ldots \\ & + f^{(p)}(x_{i-1})H_{ip0}(x) + f^{(p)}(x_i)H_{ip1}(x) \end{aligned} \tag{20.7}$$

where $H_{ijk}(x)$ is different for each subinterval.
(*i is subinterval; j is order of derivative; k is endpoint)

Using these, define:

$$p(x) = \begin{cases} p_1(x), & \text{if } x_0 \le x \le x_1 \\ \ldots & \\ p_n(x), & \text{if } x_{n-1} \le x \le x_n \end{cases} \tag{20.8}$$

For WHW#4 :
"Method 1": set up the V_i matrix for each subinterval and solve $V_i\alpha_i = f_i$ for each i.
(* f_i is the vector of data)
"Method 2": find the cardinal basis functions (the functions H_{ijk}) and ... you're done. Express each p_i like in (1.7).

Exercise: Show how to do this:
1) compute H_{ij} for $[0, 1]$
2) use change of variables to transform H_{ij} to H_{klm} for subinterval k

Quick side note about evaluating polynomials in the monomial basis.
Two options:
(e.g. $\mathbb{P}_3$)
1) $p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$
2) $p(x) = \alpha_0 + x \cdot (\alpha_1 + x \cdot (\alpha_2 + \alpha_3 x)) \Rightarrow$ Horner's rule

Exercise: Count the FLOPs needed to evaluate an n^{th} degree polynomial

$$p(x) = \alpha_0 + \alpha_1 x + ... + \alpha_n x^n \tag{20.9}$$

using both approaches.

20.4 One more method: the Newton interpolating polynomial

Idea: Use divided differences.

Definition 1. *Let $x_0, x_1, ..., x_n$ (interpolating nodes) be distinct points in $\mathbb{R}$, and let $f_0, f_1, ..., f_n$ ("the data") be function values. Then:*

$$\begin{aligned}
f[x_0] &= f(x_0) \\
f[x_0, x_1] &= \frac{f[x_0] - f[x_1]}{x_0 - x_1} \\
f[x_0, x_1, x_2] &= \frac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2} \\
&... \\
f[x_0, ..., x_k] &= \frac{f[x_0, ..., x_{k-1}] - f[x_1, ..., x_k]}{x_0 - x_k} \\
&... \\
f[x_0, ..., x_n] &= \frac{f[x_0, ..., x_{n-1}] - f[x_1, ..., x_n]}{x_0 - x_n}
\end{aligned}$$

are called divided differences.

Theorem 3. *Let $x_0, ..., x_n$ and $f_0, ..., f_n$ be as before. Let $p \in \mathbb{P}_n$ interpolate this stuff (i.e. $p(x_i) = f(x_i)$ for $i = 0, ..., n$). Then:*

$$\begin{aligned} p(x) &= f[x_0] \\ &\quad + f[x_0, x_1](x - x_0) \\ &\quad + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + ... \\ &\quad + f[x_0, ..., x_n](x - x_0)...(x - x_{n-1}) \\ &= \sum_{i=0}^{n} f[x_0, ..., x_i] \prod_{j=0}^{i-1} (x - x_j) \end{aligned} \tag{20.10}$$

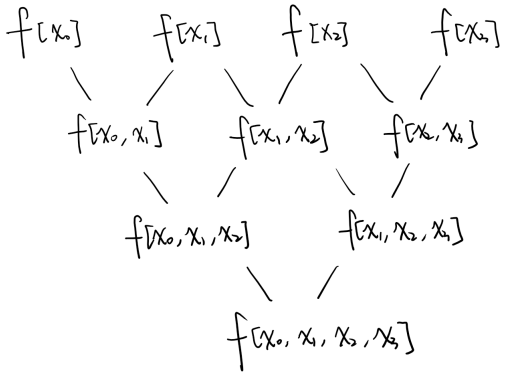
this is the Newton interpolating polynomial. (or: "the interpolating polynomial in Newton form")

(*Compute this with $p(x) = f(x_0)L_0(x) + ... + f(x_n)L_n(x)$)

Exercise: Prove this by induction. If you can't figure it out, look it up online...

Exercise: Prove that $f[x_0, ..., x_k)$ is symmetric in the arguments.

How to compute this? Picture:



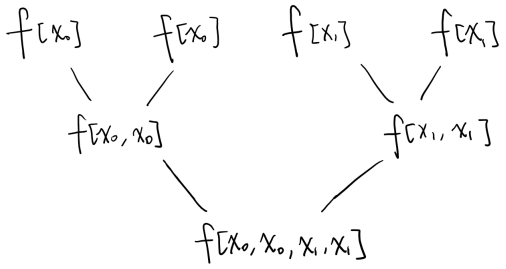
Basically: use dynamic programming.

Exercise: Compute Newton poly for (x_0, f_0), (x_1, f_1), and (x_2, f_2)

We can also do this for Hermite interpolation.
Idea: allow repeats among the x_i's.
Then: use "generalized" divided differences:

$$f[x_i, x_{i+1}, ..., x_{i+p}] = \frac{f^{(p)}(x_l)}{p!} \text{ if } x_l = x_i = ... = x_{i+p} \tag{20.11}$$

Example: for cubic Hermite:
$X : x_0, x_0, x_1, x_1$
$F : f_0, f_0', f_1, f_1'$



This works.

Exercise: Work out the example:
x_0, x_0, x_0, x_1, x_1
$f_0, f_0', f_0'', f_1, f_1'$

Chapter 21

Numerical Integration

[SFP: Scribed by Churchill Zhang.]

21.1 definition

Goal: we want to approximate the following function: $\int_A^B f(x)dx$

The occasions that we do this:

1. There is no close form expression for the anti derivative of f.

2. Do not even know f but we know what f "is" (i.e. in terms of physics, economics).

e.g. the exponential integral Ei(x).

$$Ei(x) = \int_{-x}^{+\infty} \frac{e^{-t}}{x} dt \tag{21.1}$$

and for x=1:

$$Ei(1) = \int_{1}^{+\infty} e^{-t} dt = 1.89511781.... \tag{21.2}$$

Graph:

21.2 Example

1. Error Function

$$erf(x) = \frac{x}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{21.3}$$

2. We have a quardcopter and some gyroscope measures v(t) for $t \geq 0$. We got samples of v(t) at t_j for j=0,1,2..... and $t_i = \Delta t * i$

   So to estimate $x(t_i)$, we could approximate using quadrature the integral

$$\int_{t_0}^{t_i} v(s)ds \approx \sum_{j=1}^{i} (t_j - t_{j-1})v(t_j) \tag{21.4}$$

Exercise: See if you can understand how error accumulates (bias accumulates) and (only accumulates over a short time period)
Application in solving ODEs:

$$\begin{cases} y'(t) = f(t) \\ y(0) = y_0 \end{cases} \tag{21.5}$$

Then the solution to the IVP is:

$$y(t) = y(0) + \int_0^t f(s)ds \tag{21.6}$$

Problem: Might not be able to write y(t) in closed form. Basically all numerical method for solving the equations are based on quadrature applied to $\int_0^t f(t)dt$

21.3 Quadrature Rules

1. 2^{nd} simplest example : the Trapezoid Rule ("Trapezium")

$$\int_a^b f(x)dx = \frac{b-a}{2}(f(a) + f(b)) \tag{21.7}$$

2. Another simple example: Composite Trapezoid Rule:

$$\int_{x_0}^{x_n} f(x)dx = \sum_{i=1}^{n} \frac{x_i - x_{i-1}}{2}(f(x_i) + f(x_{i-1})) \tag{21.8}$$

Thought Experiment:

There are n equal sub intervals, and we applied Trapezoid Rule to estimate $\int_a^b f(x)dx$

Q: What will effect the error range as $n \to +\infty$

1. easy case: constant function and line (y=mx+b)
2. medium case:Piece wise linear (i.e. $|x|$). Exact only if nodes of quadrature line up with the discontinuities in the derivative.

21.4 Newton-Cotes integration, "simple meta algorithm"

1. Pick Lagrange interpolation of degree n
2. Find "quadrature weight" w_i at

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i) \tag{21.9}$$

Idea: Use the degree n function f form the Lagrange interpolating polynomial:

$$P_n(x) = f(x_0)L_0(x) + ... + f(x_n)L_n(x) \tag{21.10}$$

Where L_i's are the Lagrange basis functions of order n for the interpolation nodes x_0.....x_n

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{21.11}$$

Replace f with P_n in $\int_a^b f(x)dx$ and integrate

$$\int_a^b f(x)dx \approx \int_a^b P_n(x)dx = \int_a^b \sum_{i=0}^{n} f(x_i)L_i(x)dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_i(x)dx = \sum_{i=0}^{n} w_i f(x_i) \tag{21.12}$$

Where $w_i = \int_a^b L_i(x)dx$
We could map [a,b] to [-1,1] then w depends on n and i, which could be made into a table (i.e. table for w(n,i)).

e.g. n=1, which is also Trapezoid rule for $\int_{x_0}^{x_1} f(x)dx$
Goal: To compute w_0, w_1 depending on x_0, x_1
First we need L_0, L_1 :

$$L_0(x) = \frac{x - x_1}{x_0 - x_1} \tag{21.13}$$

$$L_1(x) = \frac{x - x_0}{x_1 - x_0} \tag{21.14}$$

To compute w_0 :

$$w_0 = \int_{x_0}^{x_1} L_0(x)dx = \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1}dx = \frac{1}{x_0 - x_1}(\frac{x^2}{2} - xx_1)_{x_0}^{x_1} = -\frac{x_0 - x_1}{2} \tag{21.15}$$

Exercise. show $w_1 = \frac{x_1 - x_0}{2}$
What is this saying? →Trapezoid rule

$$\int_{x_0}^{x_1} f(x)dx \approx \frac{x_1 - x_0}{2} f(x_0) + \frac{x_1 - x_0}{2} f(x_1) = \frac{x_1 - x_0}{2}(f(x_0) + f(x_1)) \tag{21.16}$$

Exercise. Derive Simpson's Rule :
Approximate $\int_a^b f(x)dx$ using the nodes a, $\frac{a+b}{2}$, b and the degree 2 Lagrange interpolating polynomials (i.e. compute w_0, w_1, w_2 for [a,b]) to get:

$$\int_a^b f(x)dx \approx \frac{b - a}{6}(f(a) + 4f(\frac{a + b}{2}) + f(b)) \tag{21.17}$$

21.5 Cubic Hermite interpolating polynomial

$$P(x) = f(a)H_0^0(x) + f(b)H_1^0(x) + f'(a)H_0^1(x) + f'(b)H_1^1(x) \tag{21.18}$$

Follow the same procedure (replace f with p in $\int_a^b f(x)dx$ to derive the "corrected Trapezoid rule")

$$\int_a^b f(x)dx = \frac{b - a}{2}(f(a) + f(b)) - \frac{(b - a)^2}{12}(f'(b) - f'(a)) \tag{21.19}$$

21.6 error bound

Recall from lecture on Lagrange interpolation. The following error formula let $f \subset C^{n+1}([a,b])$ and let $x \subset [a,b]$.
Then there exists $\zeta(\varepsilon x_i \varepsilon) \subset (a,b)$
Such that :

$$f(x) - p_n(x) = \frac{f^{n+1}(\zeta)}{(n+1)!}\Pi_n(x) \tag{21.20}$$

Where:

$$p_n(x) = \sum_{i=0}^{n} f(x_i)L_i(x) \tag{21.21}$$

$$\Pi_n(x) = (x - x_0)(x - x_1)....(x - x_n) = \prod_{i=0}^{n}(x - x_i) \tag{21.22}$$

Define:

$$E_n(f) = \int_a^b f(x)dx - \sum_{i=0}^{n} w_i f(x_i) \tag{21.23}$$

This is the quadrature due to approximating the integral using the nth order Newton-Cotes

Theorem 4. *if* $f \subset C^{n+1}([a,b])$ *for* $n \geq 1$: $|E_n(f)| \leq \frac{\max\limits_{a\leq\zeta\leq b}|f^{n+1}(\zeta)|}{(n+1)!}\int_a^b|\Pi_n(x)|dx$(21.24)

Proof.

$$\begin{aligned}
|E_n(f)| &= |\int_a^b f(x)dx - \sum_{i=0}^{n} w_i f(x_i)| \\
&= |\int_a^b f(x) - p_n(x)dx| \\
&\leq \int_a^b |f(x) - p_n(x)|dx \\
&= \int_a^b |\frac{f^{n+1}(\zeta(x))}{(n+1)!}\Pi_0(x)|dx \\
&\leq \int_a^b \frac{\max\limits_{a\leq\zeta\leq b}|f^{n+1}(\zeta)|}{(n+1)!}\Pi_n(x)dx
\end{aligned} \tag{21.25}$$

□

As for $f(x) - p_n(x)$, there exists $\zeta = \zeta(x)$ such that

$$f(x) - p_n(x) = \frac{f^{n+a}(\zeta)}{(n+1)!}\Pi_n(x) \tag{21.26}$$

Example. Error bound for Trapezoid Rule (n=1)

$$|E_1(f)| \leq \frac{\underset{a\leq\zeta\geq b}{max}|f^2(\zeta)|}{2}\int_a^b |x-a||x-b|dx$$

$$= \frac{(b-a)^3}{12}\underset{a\leq\zeta\leq b}{max}|f''(\zeta)| \tag{21.27}$$

Exercise. Apply the theorem to show that the error incurred by approximating $\int_a^b f(x)dx$ with Simpson's rule satisfy

$$|E_2(x)| \leq \frac{(b-a)^4}{196}\underset{a<=\zeta<=b}{max}|f^3(\zeta)| \tag{21.28}$$

Exercise. Look up the error formula for Hermite interpolation and use it to derive a similar theorem. Use it to get an error bound for the corrected Trapezoid rule

Chapter 22

Generalized divided Differences

[SFP: Scribed by Panayiotis Christou, May 2nd 2022]

Programming Homework 4 Tutorial:
Professor said he thinks this homework will be the easiest yet. If you reach the end of the homework then the bonus problem should be easy to do as well so you should try it.

Goal:
The goal of the homework is to implement "sin" in Python using piecewise Hermite interpolation.

Constraint:
Implement sin only using information that you know. Meaning only using values of sin (and cos) that you know already or can calculate using trigonometry.
Why is it important that $\frac{d}{dx}sin(x) = cos(x)$?
We can look at higher order derivatives to see why:

$$\frac{d^2}{dx^2}sin(x) = -sin(x) \tag{22.1}$$

$$\frac{d^3}{dx^3}sin(x) = -cos(x) \tag{22.2}$$

$$\frac{d^4}{dx^4}sin(x) = sin(x) \tag{22.3}$$

From the higher order derivatives above we can identify the following relationship:

$$\frac{d^k}{dx^k}sin(x) = cos(x)\ for\ k = 1, 5, 9, .. \tag{22.4}$$

$$\frac{d^k}{dx^k} sin(x) = -sin(x) \; for \; k = 2, 6, 10, .. \tag{22.5}$$

$$\frac{d^k}{dx^k} sin(x) = -cos(x) \; for \; k = 3, 7, 11, .. \tag{22.6}$$

$$\frac{d^k}{dx^k} sin(x) = sin(x) \; for \; k = 4, 8, 12, .. \tag{22.7}$$

This is because the derivatives of sin have a periodicity where the derivative repeats itself every 4 derivatives i.e., the 1st derivative is the same as the 5th and the 9th, the 2nd is the same as the 6th and the 10th and so on.

Example:
If we want to do $p = 20$ order Hermite interpolation of sin on the interval $[a, b]$ then I need to know the following:

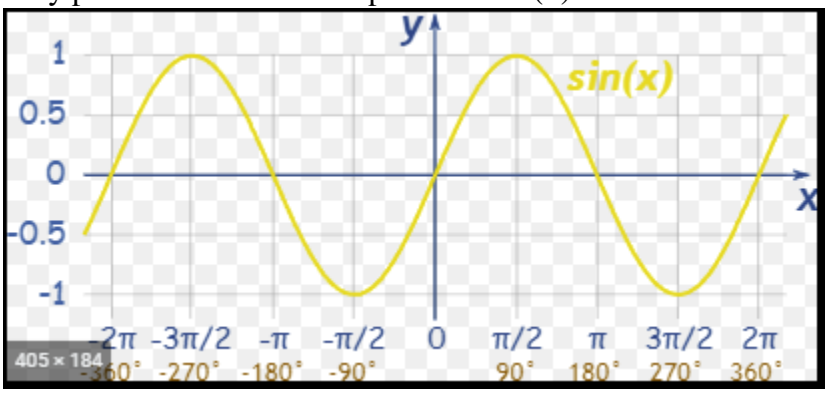
$$sin(\alpha), \frac{d}{dx} sin(\alpha), ..., \frac{d^{20}}{dx^{20}} sin(\alpha) \tag{22.8}$$

$$sin(b), \frac{d}{dx} sin(b), ..., \frac{d^{20}}{dx^{20}} sin(b) \tag{22.9}$$

To evaluate all these derivatives we just need 4 numbers:

$$sin(\alpha), sin(b), cos(\alpha), cos(b)$$

Why piecewise? Here is the period of $sin(x)$:



Hermite interpolation using the endpoints $\alpha = 0$ and $beta = 2\pi$ only is bound to fail.
So piecewise Hermite interpolation with at least 8 equal sized sub-intervals should work.
FYI: trick for question number 4 in written homework 4:
Compare V^{-1} for $[0, 2]$.

$$V^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \tag{22.10}$$

Suggestion: Use np.linalg.inv() or scipy to calculate the inverse.

Exercise:
Let $\tilde{H}_i^d(x)$ be the (ij) Cardinal basis functions for $[0, 1]$. Show that $\tilde{H}_i^d(y) = (b-a)^d \tilde{H}_i^d(\frac{y-a}{b-a})$ is the Cardinal basis function for $[a, b]$.

Method I:
Solve order p Hermite interpolation problem on $[a, b]$ by getting a to be the monomial basis vector of coefficiernts for the interpolating polynomial p. Let f be the vector of "data". We therefore have:

$$f = (f(a), f(b), f'(a), f'(b), ..., f^p(a), f^p(b))$$

To find a, we solve $Va = f$ for a.

Method II:
Let $A = V^{-1}$ where $A = [\alpha_0^0, \alpha_1^0, \alpha_0^1, \alpha_1^1, ..., \alpha_0^p, \alpha_1^p]$ is the monomial basis coefficients for the Cardinal basis functions $\tilde{H}_i^d$.
Then 'write down' p in the Cardinal basis:

$$p(x) = f(a)H_0^0(x) + f(b)H_1^0(x) + ... + f^{(p)}(a)H_0^{(p)}(x) + f^{(p)}(b)H_1^{(p)}(x)$$

OK:
So methods I and II are fine if p isn't that big. But they become very inaccurate eventually.

Why?:
The Vandermonde matrix is exponentially ill conditioned.
Condition: Idea relates to the concept of backwards stability. If you are interested look up Nick Higham's blog and find his entry on backwards stability. The condition number of a matrix is defined to be:

$$\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)} = \frac{||A||}{||A^{-1}||} -> \ e.g. for I, \ \kappa(A) = 1$$

e.g.:

$$\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow \kappa(A) = \infty$$

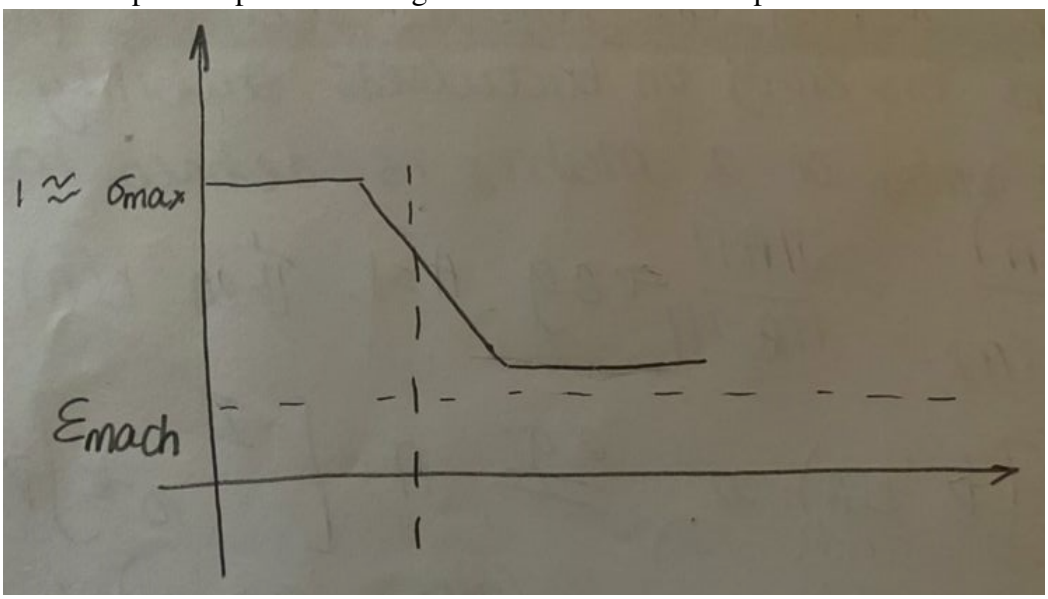
e.g.:

$$A = \begin{bmatrix} I & \cdot \\ \cdot & 10^{-k} \end{bmatrix} \Rightarrow \kappa(A) = 10^{k}$$

Idea:
The closer $\kappa(A)$ is close to 1 (note: $\kappa(A) \geq 1$), the more accurately we can solve $Ax = b$, say using LU decomposition w/ partial pivoting.

Note:
If you look in Golub and Van Loan, you will find that many of the error bounds for Gaussian elimination depends on $\kappa(A)$.
e.g. $U^T, U \in \mathbb{R}^{m \times p}, \Sigma \in \mathbb{R}^{p}, V \in \mathbb{R}^{n}$
Singular value decomposition can help us find the well-conditioned lower dimensional subspace.

One example of a plot of the singular values looks like the plot below:



Where the maximum value on the y-axis is close or equal to 1 and the minimum value is the machine ϵ and the maximum value for the x-axis is p, the number of derivatives used.

Fun Experiment:
Compare some V_s and compute their SVDs and plot their singular values like in the plot shown above. What happens?

So: Since we can't or don't want to use Methods I and II, we use Newton's interpolation (and DDs along the way).
Review (generalized) divided differences:

$$f[b,a] = \frac{f[b] - f[a]}{b - a} \; where \; f[b] = f(b) \tag{22.11}$$

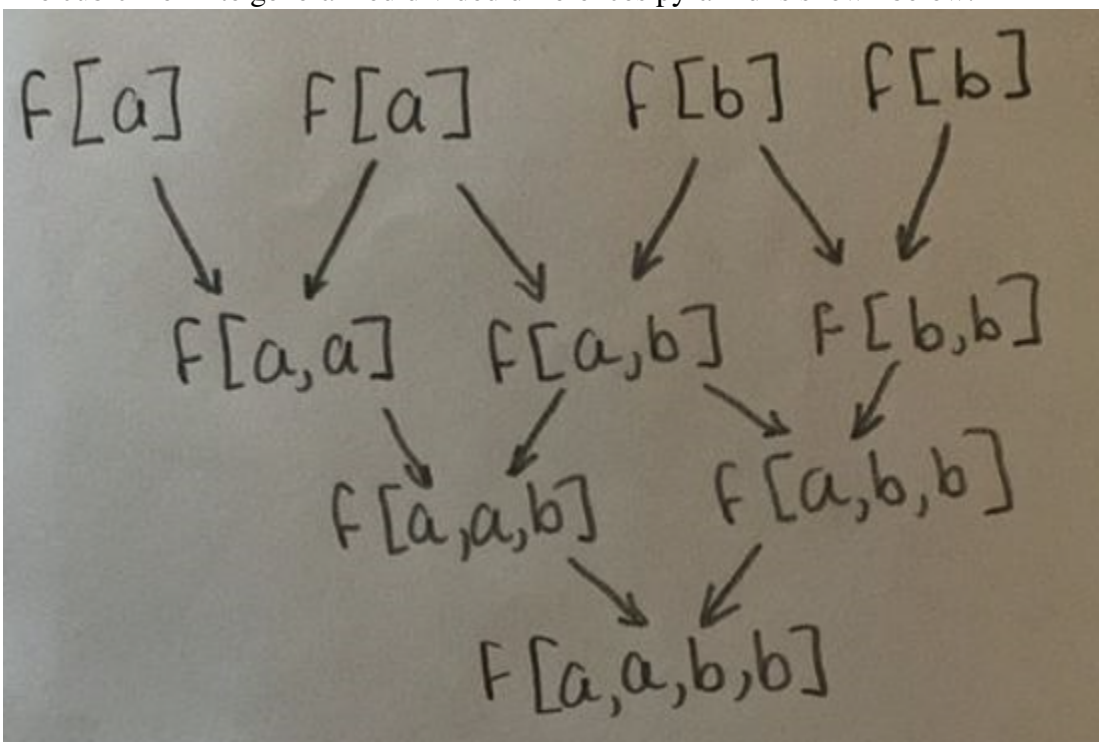
The idea was to:

1. Compute DDs
2. Write down the interpolating polynomial in Newton form

But: Since this is Hermite interpolation, we need to use generalized divided differences (GDDs). GDDs allow repeats in order to interpolate derivativs:

$$f[\alpha, \alpha, ..., \alpha] = \frac{f^{(p)}(\alpha)}{p!}$$

Where we also have p number of αs and p number of derivatives (note: including $f[\alpha]$ as the 0^{th} derivative).
e.g.
The cubic Hermite generalized divided differences pyramid is shown below:



Where the total number of numbers in the pyramid is $\frac{n(n+1)}{2}$
See below for an example of the "merge operation":

$$f[a,b,b,c]f[b,b,c,d] -> \ f[a,b,b,c,d]$$

Where the two entries in the first row each have N nodes and the resulting entry from the merging of the two previous entries has N+1 nodes.
Below we define how to compute 3 types of entries:

$$f[a] = f^0(a)/0! = f(a) \tag{22.12}$$

$$f[a,a] = f^{(1)}(a)/1! = f'(a) \tag{22.13}$$

$$f[a,b] = \frac{f[a] - f[b]}{a-b} = \frac{f(a) - f(b)}{a-b} \tag{22.14}$$

With the general case given below:

$$\frac{f[x_0,...,x_{n-1}] - f[x_1,...,x_n]}{x_0 - x_n} \tag{22.15}$$

For the homework you need the first row to look like:

$$f[a]...f[a]f[b]...f[b]$$

Where we have the first $f[a]$ denoted as x_1 and the last $f[a]$ denoted as x_{p+1} and the first $f[b]$ denoted as $p+2$ and the last $f[b]$ as $2p+2$.
Combined with the Newton form of the interpolating polynomial we have the following:

$$p(x) = f[x_1]+f[x_1,x_2](x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2)+...+f[x_1,x_2,...,x_{2p+2}](x-x_1)(x-x_2)..(x-x_{2p+2}) \tag{22.16}$$

e.g.:
Cubic Hermite divided differences:

$$p(x) = f[a] + f[a,a](x-a) + f[a,a,b](x-a)^2 + f[a,a,b,b](x-a)^2(x-b) \tag{22.17}$$

Question: What is the time complexity of computing V^{-1}?

Answer: O(p^3) where p is the number of nodes.
It is possible to solve one $V\alpha = f$ system in O(p^2) time for Lagrange interpolation, the Largange basis gives an O(p^2) algorithm.

Exercise:
What is the time complexity of computing p in Newton form? What about evaluating it?
Time complexity of computing: O(n^2)
Time complexity of evaluating: O(p)

Exercise:
What is the memory complexity? O(p^2)
Reminder:
For Lagrange interpolation:

$$x_0, x_1, ..., x_n$$

$$f_0, f_1, ..., f_n$$

And we get p as shown below:

$$p(x) = f_0 L_0(x) + ... + f_n L_n(x) \tag{22.18}$$

Final Review

[SFP: Scribed by Saif Azim.]

22.1 Finite Difference Approximation of $\frac{d^2}{dx^2}$

Let's say we have nodes $x_0 ... x_n$ where $x_{i+1} = x_i + h \rightarrow$ equally spaced
Let's say we know $f(x_0), ..., f(x_n)$, how do we approximate $f''(x_i)$ for $i = 0, 1, ..., n$?
Use taylor expansion:

$$f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2}f''(x_i) + \frac{h^3}{6}f'''(x_i) + O(h^4)$$

$$f(x_i - h) = f(x) - hf'(x_i) + \frac{h^2}{2}f''(x_i) - \frac{h^3}{6}f'''(x_i) + O(h^4)$$

adding the two:

$$f(x_i + h) + f(x_i - h) = 2f(x_i) + h^2 f''(x_i) + O(h^4)$$

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

If you approximate $f''(x_i)$ with

$$\frac{f(x_i + h) - 2f(x_i) + f(x - h)}{h^2}$$

the error is $O(h^2)$ but the error however is really

$$\frac{h^2}{12}f^{(4)}(x_i) + O(h^3)$$

22.2 Power Method Convergence

Input: Matrix A $\epsilon \mathbb{R}^{n \times n}$ A is symmetric
Output: Maximum magnitude eigenvalue of A ($\lambda_i(A)$ s.t $|\lambda_i A|$ is maximum
Assume: if $|\lambda_1| > |\lambda_2| > ....|\lambda_n|$
The Algorithm:

1. Choose $u^(0)$ s.t $||u_0||_2 = 1$
2. set $\lambda_1^0 = u_0^T A u_0$
3. for i = 1, 2, ...:

- Set $\tilde{u_1} = Au^{i-1}$
- Set $u_1^(i) = \frac{\tilde{u_1^i}}{||\tilde{u_1^i}||_2}$
- $\lambda_1^i = u_1^{iT} A u_1^i$
- if $|\lambda_1^i - \lambda_1^{i-1}| < tolerance$

Note: ignore the normalization because we can approximate eigenvalue using Rayleigh Quotient

The ith iterate of $\tilde{u_1^i}$ is just

$$A^i \tilde{u_1}^0$$

Let the Eigendecomposition of A be

$$A = Q\Lambda Q^T$$

(a) Properties of $A = Q\Lambda Q^T$

- Q is orthogonal
- $QQ^T = I = Q^T Q$
- Λ = diagonal matrix (diag($\lambda_1, ..., \lambda_n$)

$$= \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

- Because A is symmetric and Q is orthogonal, each λ_i is real

Now plug in eigenvalue decomposition into iteration

$$\tilde{u_1}^i = (Q\Lambda Q^T)\tilde{u_1}^0$$

$$Q\Lambda Q^T Q\Lambda Q^T ... Q\Lambda Q^T \tilde{u_1}^0 = Q\Lambda Q^T \tilde{u_1}^0$$

How should you interpret $Q^T \tilde{u_i}^0$? It is the vector of coeff's
$C^i = Q^T \tilde{u_1}^i$ is the vector of coeffs of $\tilde{u_1}^i$ expanded in the Q basis! It all comes back to basis'. This means that:

$$\tilde{u}^i = C_1^i q_1 + ... + C_n^i q_n = Q(Q^T \tilde{u}^i)$$

We can manipulate

$$Q^T \tilde{u_1}^i = \Lambda^i Q^T \tilde{u_i}^0$$

$$\begin{bmatrix} C_1^i \\ . \\ . \\ C_n^i \end{bmatrix} = \begin{bmatrix} \lambda_1^i & & & \\ & \lambda_2^i & & \\ & & \ddots & \\ & & & \lambda_n^i \end{bmatrix} \begin{bmatrix} C_1^0 \\ . \\ . \\ C_n^0 \end{bmatrix}$$

$$Q^T \tilde{u_i}^0 = \begin{bmatrix} \lambda_1^i c_1^0 \\ . \\ . \\ \lambda_n^i c_n^0 \end{bmatrix}$$

or

$$C_j^i = \lambda_j^i c_j^0, j = 0, 1, 2, ..., n)$$

$$\tilde{u}^i = \Sigma_{j=1}^n \lambda_j^i c_j^0 q_j$$

$$= \lambda_1^i \Sigma_{j=1}^m \frac{\lambda_j^i}{\lambda_1^i} c_j^0 q_j$$

So what happens as i goes to infinity. Hopefully we can see that we expand our vector in the q basis and taking a step in the power method raises this power on each of the lambdas, so we can see that the coeffecient for the jth q after i iterations is just λ_j^i. So we are affecting the coeffecients for each different q function independently of each other.

22.3 Singular Value Decomposition

$$A \epsilon \mathbb{R}^{mxn}, m \geq n$$

Your are the taking the maximum over unit vectors v

$$v_1 = \underset{||v||}{argmax} = 1$$

$$u_1 = \frac{u_1}{||\tilde{u_1}||}$$

define

$$\tilde{u_1} = Av_1$$

Recall that the SVD of a matrix A is:

$$A = U\Sigma V^T, p = n$$

$$U \epsilon \mathbb{R}^{mxp}, \Sigma \epsilon \mathbb{R}^{pxp}, V^T \epsilon \mathbb{R}^{nxp}$$

$$A = \Sigma_{i=1}^{p} \sigma_i u_i v_i^T$$

And the matrices look as follows:

$$U = \left[u_1 ... u_p\right], \Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}, V = \left[v_1 ... v_p\right]$$

If A is rectangular then

$$U^T U = I, V^T V = I$$

,

$$u_i^T u_j = \delta_{ij}, v_i^T v_j = \delta_{ij}$$

Furthermore:

$$u_i^T A = \Sigma_{j=1}^{p} \sigma_j u_i^T u_j v_j^T = \sigma_j v_j^T$$
$$A v_i = \Sigma_{j=1}^{p} \sigma_j u_j v_j^T v_i = \sigma_i u_i, NOTE : v_j^T v_i = \delta_{ij}$$

The idea of this proof, when you build the SVD at the end we have

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p, (\sigma_i \geq 0 \ \forall \ i)$$

You start with the biggest singular value, we try and find the u in the v pair that go along with that, and we reject them out of consideration and go down one dimension and do it again recursively until we have taken the matrix apart. So the first step in the proof/algorithm:

(a) Compute $V_1 = \frac{argmax}{||V||_2} ||Av||_2$

- We are trying to find a direction which sends the vector the furthest.

(b) then let $\sigma_1 = ||Av_1||_2$

(c) Set $u_1 = \frac{Av_1}{\sigma_1}$ (so that $\sigma_1 u_1 = Av_1$)

- The idea of the proof is that if we do this and we multiply back through and apply this step to the next lower block, that this works. The proof isnt necessarily the hardest, but does require studying a bit and finesse.

Lastly, properties of SVD are a big thing to know so note the following.

22.4 Properties of the SVD

(a) rank(A) = r = # nonzero $\sigma_i's$

(b) range(A) = $\langle u_1, \quad ... \quad , u_r \rangle$

(c) $\|A\|_2 = \sigma_1$, $\|A\|_F = \sum_{i=1}^{r} \sigma_i^2 \quad (\|A\|_F = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2)$

(d) nonzero σ_i's for A are the square root of the nonzero λ_i's of $A^T A$ or AA^T

(e) if $A = A^T$, then the σ_i's are just $|\lambda_i| \, of \, A$

(f) $|det(A)| = \prod_{i=1}^{n} \sigma_i$ if A is square