Numerical Methods

CS 357 - Fall 2016

Introduction

Numerical Methods: What?

- ‘Numerical’?
  - Has to do with (real) numbers...
  - ...in a computer
    Q: How do we even get a computer to understand real numbers?
  - ...and not just one, *arrays* of them
    Q: What’s an ‘array’? How does a computer deal with it?
- ‘Method’?
  - Think ‘algorithm’. But there’s more:
    - The algorithm comes from a math idea
    - For each math idea, there are lots of algorithms
    - Some fast, some slow
    - Some accurate, some inaccurate

Wait–how did ‘accuracy’ come into this?

- Math + Complexity + Accuracy = Method

Accuracy

- Why might a numerical method **not give the right answer?** (i.e. be inaccurate)
  - Because (unlike in the special cases that math has taught you), mostly we *can't write down the answer*. Not in a finite amount of space anyway. And a computer *is* finite.

Demo: Waiting for 1

Numerical Experiments

Model:

- Small-scale behavior easy to describe
- Large-scale behavior desired, but hard to understand

Demo: Brownian Motion

Numerical Experiments

- What are we going to want to know about a numerical experiment?
  - What question are we attempting to answer?
  - What is the outcome of the experiment?
    What does it predict?
  - Is the answer accurate? Does it match the question?
  - How long will it take?
  - **Better:** How long until we have an acceptable answer?
    Observation: Time-accuracy trade-off
  - Is the experiment repeatable?
  - Efficient: Is running this a good use of our time/computer?

Class web page

`bit.ly/cs357-f16`

- Assignments
  - HW0!
  - Pre-lecture quizzes
  - In-lecture interactive content (bring computer or phone if possible)
- Exams
- Class outline q(with links to notes/demos/activities/quizzes)
- Scribbles
- Virtual Machine Image
- Piazza

- Policies
- Video
- Interactive Questions
- Calendar
    - Office Hours

In-class activity: Complexity of Matrix-Matrix Multiplication

Recap: Understanding Asymptotic Behavior, $O(\cdot)$ *Notation*

Demo: Cost of Matrix-Matrix Multiplication

- Can we say anything exact about our results?
  - Observed: Time for $n = 800$ was about $8\times$ that for $n = 400$
  - Does a linear model fit? $\text{Time} \approx c \cdot n$?
  - Does a quadratic model fit? $\text{Time} \approx c \cdot n^2$?
  - Does a cubic model fit? $\text{Time} \approx c \cdot n^3$? Yep.
  - **Problem:** Still not necessarily valid for each individual value.
- How do we say something exact without having to predict individual values exactly?

  Solution: $O(\cdot)$ notation

  Idea: Let $g(n)$ be our 'model function' ($g(n) = n^3$ above)

Then: Say

$$\mathrm{Time}(n) = O(g(n))$$

to mean: There is a constant C so that

$$\mathrm{Time}(n) \leqslant C \cdot g(n).$$

Assume Time(n) non-negative, otherwise add absolute values.

Important: Not just time: also errors, growth, ...

Making Predictions with $O(\cdot)$-Notation

- Suppose you know that $\text{Time}(n) = O(n^2)$. And you know that for $n_1 = 1000$, the time taken was 5 seconds. Estimate how much time would be taken for $n_2 = 2000$.

$$\text{Time}(n_1) \approx C \cdot n_1^2 = 5$$

Could use that to find coefficient C. Or: just use the ratio.

$$\text{Time}(n_2) \approx C \cdot n_2^2 = C \cdot \left(\frac{n_2}{n_1}\right)^2 n_1^2 = \left(\frac{n_2}{n_1}\right)^2 \cdot \text{Time}(n_1) = 2^2 \cdot 5\,\text{s} = 20\,\text{s}.$$

Part 1:
Models, Errors, and Numbers

1 Python, Numpy, and Matplotlib

Programming Language: Python/numpy

- Reasonably readable
- Reasonably beginner-friendly
- Mainstream (top 5 in 'TIOBE Index')
- Free, open-source
- Great tools and libraries (not just) for scientific computing
- Python 2/3? 3!
- `numpy`: Provides an array datatype
  Will use this and `matplotlib` all the time.
- See class web page for learning materials

- **Demo:** Python
- **Demo:** numpy
- **In-class activity:** Image Processing

2 Making Models with Polynomials

Why polynomials?

$$a_3x^3 + a_2 x^2 + a_1 x + a_0$$

- How do we write the general case?

$$\sum_{i=1}^{n} a_i x^i.$$

- *Why* polynomials and not something else?
  - We can add, multiply, maybe divide (grade school, computer HW)
  - More complicated functions (e^x, $\sin x$, $\sqrt{x}$) have to be *built* from those parts $\rightarrow$ at least approximately
  - Easy to work with as a building block.
    General recipe for numerics: Approximate *thing* with a polynomial, perform operation on that, evaluate. (e.g. calculus, root finding)

Reconstructing a Function From Derivatives

- If we know $f(x_0)$,$f'(x_0)$,$f''(x_0)$, can we approximately reconstruct the function as a polynomial p?

$$p(x) = ??? + ???x + ???\,x^2 + \cdots$$

Well, let's try. Set $x_0 = 0$ for simplicity and assume (for example, for degree four):

$$p(x) = a + b\,x + c\,x^2 + d\,x^3 + e\,x^4$$

$p(x_0) = f(x_0)$, i.e. $p(0) = f(0)$ yields $a = f(0)$. Take a derivative:

$$f'(x) = b + 2c\,x + 3d\,x^2 + 4\,e\,x^3$$

$p'(0) = f'(0)$ yields $b = f'(0)$.

$$f''(x) = 2c + 3 \cdot 2d\,x + 4 \cdot 3e\,x^2$$

$p''(0) = f''(0)$ yields $c = f''(0)/2$.

$$f'''(x) = 3 \cdot 2d + 4 \cdot 3 \cdot 2ex$$

$p'''(0) = f'''(0)$ yields $d = f'''(0)/3!$. (and so on)

Found: Taylor series approximation.

$$f(0 + x) \approx f(0) + f'(0)x + \frac{f''(0)}{2}x^2 + \cdots$$

General pattern:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!}x^i$$

Demo: Polynomial Approximation with Derivatives (Part I)

Shifting the Expansion Center

- Can you do this at points other than the origin?

  In this case, 0 is the center of the series expansion. Can also shift to another center x_0–using the 'substitution':

$$
\begin{aligned}
0 &\mapsto x_0 \\
x &\mapsto x - x_0
\end{aligned}
$$

$$f(x) = f(x_0 + (x - x_0)) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \cdots$$

  Since there's a lot of $x - x_0$ going around here, rewrite again using $h = x - x_0$ to shorten:

$$f(x_0 + h) \approx f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2}h^2 + \cdots,$$

or, generally,

$$f(x_0 + h) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_0)}{i!} h^i.$$

Errors in Taylor Approximation (I)

- Can't sum infinitely many terms. Have to truncate. How big of an error does this cause?

  Demo: Polynomial Approximation with Derivatives (Part II)

  Suspicion:

$$\underbrace{\left| f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right|}_{\text{Taylor error for degree } n} \leqslant C \cdot h^{n+1}$$

  or

$$\underbrace{\left| f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right|}_{\text{Taylor error for degree } n} = O(h^{n+1}).$$

  Turns out: Actually true. Will see in a little while.

Making Predictions with Taylor Truncation Error

- Suppose you expand $\sqrt{x-10}$ in a Taylor polynomial of degree 3 about the center $x_0 = 12$. For $h_1 = 0.5$, you find that the Taylor truncation error is about 10^{-4}.

  What is the Taylor truncation error for $h_2 = 0.25$?

  $\text{Error}(h) = O(h^{n+1})$, where $n = 3$, i.e.

  $$\text{Error}(h_1) \approx C \cdot h_1^4$$

  $$\text{Error}(h_2) \approx C \cdot h_2^4$$

  Again: Use the ratio of h_2 / h_1.

  $$\text{Error}(h_2) \approx C \cdot h_2^5 = C \cdot \left(\frac{h_2}{h_1}\right)^4 h_1^5 \approx \left(\frac{h_2}{h_1}\right)^4 \cdot \text{Error}(h_1).$$

  Can make prediction of the error for one h if we know another.

Demo: Polynomial Approximation with Derivatives (Part III)

Taylor Remainders: the Full Truth

Let $f\colon \mathbb{R} \to \mathbb{R}$ be $n+1$-times differentiable on the interval (x_0, x) with $f^{(n)}$ continuous on $[x_0, x]$. Then there exists a $\xi \in (x_0, x)$ so that

$$f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i = \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!}}_{\text{“}C\text{”}} \cdot (\xi - x_0)^{n+1}$$

and since $|\xi - x_0| \leqslant h$

$$\left| f(x_0+h) - \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!} h^i \right| \leqslant \underbrace{\frac{|f^{(n+1)}(\xi)|}{(n+1)!}}_{\text{“}C\text{”}} \cdot h^{n+1}.$$

In-class activity: Taylor series

Using Polynomial Approximation

- Suppose we can approximate a function as a polynomial:

$$f(x) \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

How is that useful? Say, if I wanted the integral of f?

Easy: Just integrate the polynomial:

$$\begin{aligned} \int_a^b f(x)\mathrm{d}x &\approx \int_a^b a_0 + a_1 x + a_2 x^2 + a_3 x^3 \mathrm{d}x \\ &= a_0 \int_a^b 1\mathrm{d}x + a_1 \int_a^b x \cdot \mathrm{d}x + a_2 \int_a^b x^2 \mathrm{d}x + a_3 \int_a^b x^3 \mathrm{d}x \end{aligned}$$

Even if you had *no idea* how to integrate f using calculus, you can *approximately* integrate f anyway, by taking a bunch of derivatives (and forming a Taylor polynomial).

Demo: Computing π with Taylor

Reconstructing a Function From Point Values

- If we know function values at some points $f(x_1), f(x_2), \ldots, f(x_n)$, can we reconstruct the function as a polynomial?

$$f(x) = ??? + ???x + ???x^2 + \cdots$$

Instead, just write down what we want:

$$\begin{aligned} a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \cdots &= f(x_1) \\ \vdots \quad & \quad \vdots \\ a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots &= f(x_n) \end{aligned}$$

Q: How many a_i can we (uniquely) determine this way? (Answer: n–same number of unknowns as equations.)
So:

$$\begin{aligned} a_0 + a_1 \cdot x_1 + a_2 \cdot x_1^2 + \cdots + a_{n-1} \cdot x_1^{n-1} &= f(x_1) \\ \vdots \quad & \quad \vdots \end{aligned}$$

$$a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots + a_{n-1} \cdot x_n^{n-1} \ = \ f(x_n)$$

Rewrite that in matrix form:

$$\underbrace{\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}}_{\boldsymbol{a}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}}_{\boldsymbol{f}}.$$

Can solve linear system to find $(a_0, \ldots, a_n)$. Then: have interpolant

$$\tilde{f}(x) = a_0 + a_1 \cdot x_n + a_2 \cdot x_n^2 + \cdots + a_{n-1} \cdot x_n^{n-1}$$

that obeys $\tilde{f}(x_i) = f(x_i)$.
Super important. Will see this a *lot*.

V is called the Vandermonde matrix.

Main lesson:

$$V(\text{coefficients}) = (\text{values at points}).$$

This is called interpolation. (x_i) are called the nodes.

Some slick formulas (like Taylor) exist, too.

Surprisingly 'interesting' numerical properties.
$\rightarrow$ What that means is that this may *look* bulletproof. But it can break in a number of 'interesting' ways. Will learn more about those later.

Demo: Polynomial Approximation with Point Values

Error in Interpolation

- What did we (empirically) observe about the error in interpolation in the demo?

  To fix notation: f is the function we're interpolating. $\tilde{f}$ is the interpolant that obeys $\tilde{f}(x_i) = f(x_i)$ for $x_i = x_1 < \ldots < x_n$. Let $h = x_n - x_1$ be the interval length.

  For $x \in [x_1, x_n]$:

  $$|f(x) - \tilde{f}(x)| = O(h^n)$$

  Can predict errors with this just like for Taylor.

- What is the error *at* the interpolation nodes?

  Zero–we're matching the function exactly there.

- Care to make an unfounded prediction? What will you call it?

It looks like approximating by an $(n-1)$th degree polynomial somewhat generally results in an $O(h^n)$ error. This is called convergence of order n or nth order convergence.

Making Use of Interpolants

- Suppose we can approximate a function as a polynomial:

$$f(x) \approx a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

How is that useful? Say, if I wanted the integral of f?

Easy: Just integrate the interpolant:

$$\begin{aligned} \int_a^b f(x)\mathrm{d}x &\approx \int_a^b a_0 + a_1 x + a_2 x^2 + a_3 x^3 \mathrm{d}x \\ &= a_0 \int_a^b 1\mathrm{d}x + a_1 \int_a^b x \cdot \mathrm{d}x + a_2 \int_a^b x^2 \mathrm{d}x + a_3 \int_a^b x^3 \mathrm{d}x \end{aligned}$$

Even if you had *no idea* how to integrate f using calculus, you can *approximately* integrate f anyway, by taking a bunch of function values (and forming an interpolant).

Demo: Computing π with Interpolation

More General Functions

- Is this technique limited to the monomials $\{1, x, x^2, x^3, \ldots\}$?

  No, not at all. Works for any set of functions $\{\varphi_1, \ldots, \varphi_n\}$ for which the generalized Vandermonde matrix

$$\begin{pmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_n(x_n) \end{pmatrix}$$

  is invertible.

Interpolation with General Sets of Functions

For a general set of functions $\{\varphi_1, \ldots, \varphi_n\}$, solve the linear system with the generalized Vandermonde matrix for the coefficients $(a_1, \ldots, a_n)$:

$$\underbrace{\begin{pmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_n(x_n) \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}}_{\boldsymbol{a}} = \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}}_{\boldsymbol{f}}.$$

- Given those coefficients, what is the interpolant $\tilde{f}$ satisfying $\tilde{f}(x_i) = f(x_i)$?

$$\tilde{f}(x) = \sum_{i=1}^{n} a_i \varphi_i(x).$$

3 Making Models with Monte Carlo

Randomness: Why?

- What types of problems can we solve with the help of random numbers?

  We can compute (potentially) *complicated averages*.

  - Where does 'the average' web surfer end up? (PageRank)
  - How much is my stock portfolio/option going to be worth?
  - How will my robot behave if there is measurement error?

Random Variables

- What is a random variable?

  A random variable X is a function that depends on 'the (random) state of the world'.

  Example: X could be

  - 'how much rain tomorrow?', or
  - 'will my buttered bread land face-down?'

  Idea: Since I don't know the entire state of the world (i.e. all the influencing factors), I can't know the value of X.

  $\rightarrow$ Next best thing: Say something about the *average* case.

  To do that, I need a probability distribution for the values of X.

- discrete distribution:

Event	$X = x_1$	$X = x_2$	$\cdots$	$X = x_n$
Probability	p_1	p_2	$\cdots$	p_n

Need: $p_i \geqslant 0$ for the word 'probability' to make sense.

- continuous distribution:
  - Values are arbitrary real numbers
  - Each individual value has *zero probability*
  - *But:* Ranges 'value in range/interval $[a, b]$' has non-zero probability $\int_a^b p(x) \mathrm{d}x$ where p is the probability density.

Need: $p(x) \geqslant 0$ for 'probability' to make sense.

Demo: Plotting Distributions with Histograms

Expected Values/Averages: What?

- Define 'expected value' of a random variable.

  For a discrete random variable X:

  $$E[f(X)] = \sum_{i=1}^{n} p_i f(x_i)$$

  For a continuous random variable:

  $$E[f(X)] = \int_{\mathbb{R}} f(x) \cdot p(x) \mathrm{d}x$$

- Define variance of a random variable.

  $$\sigma^2[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2.$$

  'Average squared distance from the average'

Expected Value: Example I

- What is the expected snowfall in Champaign?

$$E[\text{Snow}] = \int_{\text{day} \in \text{year}} \text{Snow}(\text{day}) \cdot p(\text{day}) d_{\text{day}}?$$

What's Snow(day)? $p(\text{day})$?

- Snow(day): How much snow fell in Champaign on a given day.
- $p(\text{day})$: If I picked any day at random, how likely would each one be?

  $p(\text{day}) = \frac{1}{365} \rightarrow$ uniformly distributed.

Expected Value: Example II

- What is the expected snowfall in Illinois?

$$E[\text{Snow}] = \int_{\mathbb{R}}\int_{\mathbb{R}} \text{Snow}(x, y) \cdot p(x, y) \mathrm{d}x\mathrm{d}y?$$

What's $p(x, y)$? What's $\text{Snow}(x, y)$?

  - Snow(x,y): How much snow fell at longitude x and latitude y.
  - $p(x, y)$: Probability that a point with longitude x and latitude y is in Illinois.

Can check whether (x, y) is in Illinois. But: Need p to be a probability density, i.e.

$$\int_{\mathbb{R}}\int_{\mathbb{R}} p(x, y) \mathrm{d}x\mathrm{d}y = 1.$$

$$p(x) = \frac{1}{\int_{\mathbb{R}}\int_{\mathbb{R}} p(x, y) \mathrm{d}x\mathrm{d}y} \cdot \begin{cases} 1 & \text{if } (x, y) \text{ is in Illinois} \\ 0 & \text{if it's not.} \end{cases}$$

Working towards computing expected values $\rightarrow$ can use that twice here:

- Once to compute the normalization factor.
  $\int_{\mathbb{R}} \int_{\mathbb{R}} p(x, y) \mathrm{d}x \mathrm{d}y \sim E[1]$, up to a constant factor.
- And once more to compute $E[\text{Snow}]$.

Tool: Law of Large Numbers

Terminology:

- *Sample*: A random number x_i whose values follow a distribution $p(x)$.

In words:

- As the number of samples $N \to \infty$, the average of samples converges to the expected value with probability 1.

In symbols:

$$P\left[\lim_{N \to \infty} \frac{1}{N}\left(\sum_{n=1}^{N} x_i\right) = E[X]\right] = 1.$$

Or:

$$E[X] \approx \frac{1}{N}\left(\sum_{n=1}^{N} x_i\right)$$

Sampling: Approximating Expected Values

Integrals and sums in expected values are often challenging to evaluate.

- How can we approximate an expected value?

  Idea: Draw random samples. Make sure they are distributed according to $p(x)$.

  1. *Draw N samples* x_i distributed according to $p(x)$.
  2. Approximate
  $$E[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i).$$

- What is a Monte Carlo method?

  One in which the computed approximate result is *random*.
  Sampling is a MC method.

Expected Values with Hard-to-Sample Distributions

- Computing the sample mean requires samples from the distribution $p(x)$ of the random variable X. What if such samples aren't available?

  Find a different random variable $\tilde{X}$ with distribution $\tilde{p}(x)$ so that $\tilde{p}(x) \neq 0$ if $p(x) \neq 0$. Then:

$$
\begin{aligned}
E[X] &= \int_{\mathbb{R}} x \cdot p(x) \mathrm{d}x \\
&= \int_{\mathbb{R}} x \cdot \frac{p(x)}{\tilde{p}(x)} \tilde{p}(x) \mathrm{d}x \\
&= E\left[\tilde{X} \cdot \frac{p(\tilde{X})}{\tilde{p}(\tilde{X})} \right].
\end{aligned}
$$

  (Discrete case goes analogously.)

Switching Distributions for Sampling

Found:

$$E[X] = E\left[\tilde{X} \cdot \frac{p(\tilde{X})}{\tilde{p}(\tilde{X})}\right]$$

- How do we apply this for sampling?

  Starting point: X is hard to sample from, $\tilde{X}$ is easy to sample from (think uniform). Both have known distribution functions $p(x)$ and $\tilde{p}(x)$.

  Then we can approximate $E[X]$ bys sampling from $\tilde{X}$:

$$E[X] \approx \frac{1}{N}\sum_{i=1}^{N} \tilde{x_i} \cdot \frac{p(\tilde{x_i})}{\tilde{p}(\tilde{x_i})}.$$

- When is this a good way to sample?

When $p \approx \tilde{p}$, because then each sample contributes (roughly) equally.

Dealing with Unknown Scaling

- What if a distribution function is only known up to a constant factor, e.g.

$$p(x) = C \cdot \underbrace{\begin{cases} 1 & \text{point } x \text{ is in IL,} \\ 0 & \text{it isn't.} \end{cases}}_{\hat{p}(x)}$$

Typically $\int_{\mathbb{R}} \hat{p} \neq 1$. We need to find C so that $\int p = 1$, i.e.

$$C = \frac{1}{\int_{\mathbb{R}} \hat{p}(x) \mathrm{d}x}.$$

Idea: Use sampling.

Need to write $\int \hat{p}$ as an expected value.

$$\int_{\mathbb{R}} \hat{p}(x) \mathrm{d}x = \int_{\mathbb{R}} \frac{\hat{p}(x)}{\tilde{p}(x)} \tilde{p}(x) \mathrm{d}x = E\left[\frac{\hat{p}(\tilde{X})}{\tilde{p}(\tilde{X})}\right] \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\hat{p}(\tilde{x}_i)}{\tilde{p}(\tilde{x}_i)}$$

where $\bar{X}$ is distributed according to $\bar{p}$. For uniformly distributed $\bar{p}$ on the interval $[a, b]$, this simply becomes:

$$\int_{\mathbb{R}} \hat{p}(x)\mathrm{d}x \approx \frac{b-a}{N}\sum_{i=1}^{N} \hat{p}(\tilde{x}_i).$$

Demo: Computing π using Sampling

Demo: Errors in Sampling

Sampling: Error

The Central Limit Theorem states that with

$$S_n := x_1 + x_2 + \cdots + x_n$$

for the (x_i) independent and identically distributed we have that

$$\frac{S_n - n\,E[x_i]}{\sqrt{\sigma^2[x_i]n}} \to \mathcal{N}(0,1),$$

i.e. that term approaches the normal distribution. Or, short and imprecise:

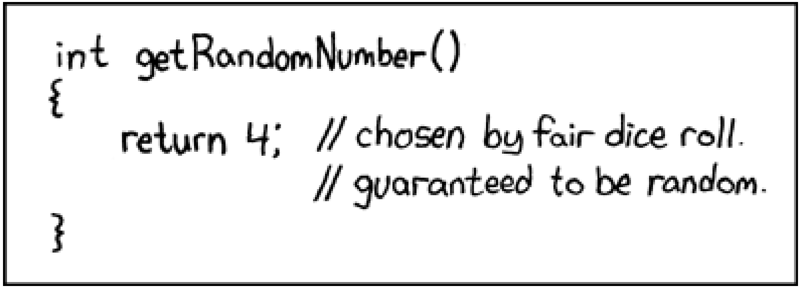
$$\left|\frac{1}{n}S_n - E[x_i]\right| = O\left(\frac{1}{\sqrt{n}}\right).$$

In-class activity: Monte-Carlo Methods

Monte Carlo Methods: The Good and the Bad

- What are some *advantages of MC methods?*
  - Computes integrals when nothing else will
  - Convergence does not depend on dimensionality
  - Still applies when deterministic modeling fails
- What are some *disadvantages* of MC methods?
  - Convergence is very slow ($O(1/\sqrt{n})$)
  - Outcome is non-deterministic

Computers and Random Numbers



[from xkcd]

- How can a computer make random numbers?

  It kind of can't. Computers are predictable. Random numbers aren't supposed to be.

  Option 1: Stick an *source of actual randomness* into the computer.

  - Don't have to look very far: Arrival times of network packets, mouse movements, ... are all sort of random.
  - `xxd /dev/random`

- `xxd /dev/urandom`
  Difference?
- ~40 bucks will buy you one: e.g. Altus Metrum ChaosKey

Waaay too much effort that way.

Random Numbers: What do we want?

- What properties can 'random numbers' have?
    - Have a specific distribution
      (often 'uniform'–each value between, say, 0 and 1, is equally likely)
    - Real-valued/integer-valued
    - Repeatable (i.e. you may *ask* to exactly reproduce a sequence)
    - Unpredictable
        - V1: 'I have no idea what it's going to do next.'
        - V2: No amount of engineering effort can get me the next number.
    - Uncorrelated with later parts of the sequence
      (Weaker: Doesn't repeat after a short time)
    - Usable on parallel computers

What's a Pseudorandom Number?

- Actual randomness seems like a lot of work. How about 'pseudorandom numbers?'

  Idea: Maintain some 'state'. Every time someone asks for a number:

  $$\text{random\_number}, \text{new\_state} = f(\text{state})$$

  Satisfy:

  - Distribution
  - 'I have no idea what it's going to do next.'
  - Repeatable (just save the state)
  - Typically *not* easy to use on parallel computers

Demo: Playing around with Random Number Generators

Some Pseudorandom Number Generators

Lots of variants of this idea:

- LC: 'Linear congruential' generators
- MT: 'Mersenne twister'

Remarks:

- Initial state and parameter choice often surprisingly tricky.
  Bad choice: Predictable/correlated numbers.
  E.g. Debian OpenSSL RNG disaster
- Absolutely **no reason** to use LC or MT any more. (Although almost all randonumber generators you're likely to find are based on those–Python's `random` module, `numpy.random`, C's `rand()`, C's `rand48()`.
- These are **obsolete.**

Counter-Based Random Number Generation (CBRNG)

- What's a CBRNG?

  Idea: Cryptography has *way* stronger requirements than RNGs.
  And the output *must* 'look random'.

  E.g. AES: 128 encrypted bits=AES(128-bit plaintext, 128 bit key)

  Read that as: 128 random bits=AES(128-bit counter, arbitrary 128 bit key)

  - Just use $1, 2, 3, 4, 5, \ldots.$ as the counter.
  - *No* quality requirements on counter or key to obtain high-quality random numbers
  - *Very* easy to use on parallel computers
  - Often accelerated by hardware, faster than the competition

Demo: Counter-Based Random Number Generation

4 Error, Accuracy and Convergence

Error in Numerical Methods

- Every result we compute in Numerical Methods is inaccurate. What is our model of that error?

$$\text{Approximate Result} = \text{True Value} + \text{Error}.$$

$$\tilde{x} = x_0 + \Delta x.$$

- Suppose the true answer to a given problem is x_0, and the computed answer is $\tilde{x}$. What is the absolute error?

  $|x_0 - \tilde{x}|$.

- What is the relative error?

$$\frac{|x_0 - \tilde{x}|}{|x_0|}$$

- Why introduce relative error?

  Because absolute error can be misleading, depending on the magnitude of x_0. Take an absolute error of 0.1 as an example.

  - If $x_0 = 10^5$, then $\tilde{x} = 10^5 + 0.1$ is a fairly accurate result.
  - If $x_0 = 10^{-5}$, then $\tilde{x} = 10^{-5} + 0.1$ is a completely inaccurate result.

  Relative error is independent of magnitude.

- What is meant by 'the result has 5 accurate digits'?

  Say we compute an answer that gets printed as

$$3.1415777777.$$

  The closer we get to the correct answer, the more of the leading digits will be right:

$$3.1415777777.$$

This result has 5 accurate digits. Consider another result:

$$123,477.7777$$

This has four accurate digits. To determine the number of accurate digits, start counting from the front (most-significant) non-zero digit.

Observation: 'Accurate digits' is a measure of relative error.

'$\tilde{x}$ has n accurate digits' is roughly equivalent to having a relative error of 10^{-n}.

$$\frac{|\tilde{x} - x_0|}{|x_0|} < 10^{-n}.$$

Measuring Error

- Why is $|\tilde{x}| - |x_0|$ wrong and a terrible measure of the error?

  Because it would claim that $\tilde{x} = -5$ and $x_0 = 5$ have error 0.

- If $\tilde{\boldsymbol{x}}$ and $\boldsymbol{x}_0$ are vectors, how do we measure the error?

  Using something called a vector norm. Will introduce those soon. Basic idea: Use norm in place of absolute value. Symbol: $\|\boldsymbol{x}\|$. E.g. for relative error:

$$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}_0\|}{\|\boldsymbol{x}_0\|}.$$

Sources of Error

- What are the main sources of error in numerical computation?
  - Truncation error:
    (E.g. Taylor series truncation, finite-size models, finite polynomial degrees)
  - Rounding error
    (Numbers only represented with up to ~15 accurate digits.)

Digits and Rounding

- Establish a relationship between '*accurate digits*' and rounding error.

  Suppose a result gets rounded to 4 digits:

  $$3.1415926 \quad \to \quad 3.142.$$

  Since computers always work with finitely many digits, they must do something similar. By doing so, we've introduced an error–'rounding error'.

  $$|3.1415926 - 3.142| = 0.0005074$$

  Rounding to 4 digits leaves 4 accurate digits–a relative error of 10^{-4}.

  Computers round *every* result–so they *constantly* introduce relative error. (Will look at how in a second.)

Condition Numbers

- Methods f take input x and produce output $y = f(x)$.
  Input has (relative) error $|\Delta x| / |x|$.
  Output has (relative) error $|\Delta y| / |y|$.
  Q: Did the method make the relative error bigger? If so, by how much?

  The condition number provides the answer to that question.
  It is simply the smallest number κ across all inputs x so that

  $$\text{Rel error in output} \leqslant \kappa \cdot \text{Rel error in input},$$

  or, in symbols,

  $$\kappa = \max_x \frac{\text{Rel error in output } f(x)}{\text{Rel error in input } x} = \max_x \frac{\frac{|f(x) - f(x + \Delta x)|}{|f(x)|}}{\frac{|\Delta x|}{|x|}}.$$

nth-Order Accuracy

Often, *truncation error* is controlled by a parameter h.

Examples:

- distance from expansion center in Taylor expansions
- length of the interval in interpolation

A numerical method is called 'nth-order accurate' if its truncation error $E(h)$ obeys

$$E(h) = O(h^n).$$

5 Floating Point

Wanted: Real Numbers... in a computer

- Computers can represent *integers*, using bits:

$$23 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (10111)_2$$

How would we represent fractions, e.g. 23.625?

Idea: Keep going down past zero exponent:

$$23.625 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

So: Could store

- a fixed number of bits with exponents $\geqslant 0$
- a fixed number of bits with exponents < 0

This is called fixed-point arithmetic.

Fixed-Point Numbers

- Suppose we use units of 64 bits, with 32 bits for exponents $\geqslant 0$ and 32 bits for exponents <0. What numbers can we represent?

2^{31}	$\cdots$	2^0	2^{-1}	$\cdots$	2^{-32}

  Smallest: $2^{-32} \approx 10^{-10}$
  Largest: $2^{31} + \cdots + 2^{-32} \approx 10^9$

- How many 'digits' of relative accuracy (think relative rounding error) are available for the smallest vs. the largest number?

  For large numbers: about 19
  For small numbers: few or none

  Idea: Instead of *fixing* the location of the 0 exponent, let it float.

Floating Point numbers

- Convert $13 = (1101)_2$ into floating point representation.

$$13 = 2^3 + 2^2 + 2^0 = (1.101)_2 \cdot 2^3$$

- What pieces do you need to store an FP number?

  Significand: $(1.101)_2$
  Exponent: 3

  Idea: Notice that the leading digit (in binary) of the significand is always one.

  Only store '101'. Final storage format:

Significand: 101 – a fixed number of bits
Exponent: 3 – a (*signed!*) integer allowing a certain range

Exponent is most often stored as a positive 'offset' from a certain negative number. E.g.

$$3 = \underbrace{-1023}_{\text{implicit offset}} + \underbrace{1026}_{\text{stored}}$$

Actually stored: 1026, a positive integer.

In-class activity: Floating Point

Unreplesentable numbers?

- Can you think of a somewhat central number that we cannot represent as

$$x = (1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)_2 \cdot 2^{-p}?$$

  Zero. Which is somewhat embarrassing.

  Core problem: The implicit 1. It's a great idea, were it not for this issue.

  Have to break the pattern. **Idea:**

  - Declare one exponent 'special', and turn off the leading one for that one.
    (say, -1023, a.k.a. stored exponent 0)
  - For all larger exponents, the leading one remains in effect.

  Bonus Q: With this convention, what is the binary representation of a zero?

Demo: Picking apart a floating point number

Subnormal Numbers

- What is the smallest representable number in an FP system with 4 stored bits in the significand and an exponent range of $[-7, 7]$?

  First attempt:

  - Significand as small as possible $\to$ all zeros after the implicit leading one
  - Exponent as small as possible: -7

  So:

  $$(1.0000)_2 \cdot 2^{-7}.$$

  Unfortunately: wrong. We can go way smaller by using the special exponent (which turns off the implicit leading one). We'll assume that the special exponent is -8. So:

  $$(0.0001)_2 \cdot 2^{-8}$$

Numbers with the special epxonent are called subnormal (or denormal) FP numbers. Technically, zero is also a subnormal.

Note: It is thus quite natural to 'park' the special exponent at the low end of the exponent range.

Note: Why would you want to know about subnormals? Because computing with them is often slow, because it is implemented using 'FP assist', i.e. not in actual hardware. Many C compilers support options to 'flush subnormals to zero'.

- FP systems without subnormals will underflow (return 0) as soon as the exponent range is exhausted.
- This smallest representable *normal* number is called the underflow level, or UFL.

- Beyond the underflow level, subnormals provide for gradual underflow by 'keeping going' as long as there are bits in the significand, but it is important to note that subnormals don't have as many accurate digits as normal numbers.
- Analogously (but much more simply–no 'supernormals'): the overflow level, OFL.

Demo: Density of Floating Point Numbers
Demo: Floating Point vs. Program Logic

Floating Point and Rounding Error

What is the relative error produced by working with floating point numbers?

- What is smallest floating point number > 1? Assume 4 stored bits in the significand.

$$(1.0001)_2 \cdot 2^0 = x \cdot (1 + 0.0001)_2$$

- What's the smallest FP number > 1024 in that same system?

$$(1.0001)_2 \cdot 2^{10} = x \cdot (1 + 0.0001)_2$$

- Can we give that number a name?

  Unit roundoff or machine precision or machine epsilon or $\varepsilon_{\text{mach}}$ is the smallest number such that

$$\text{float}(1 + \varepsilon) > 1.$$

Ignoring possible subtleties about rounding, in the above system, $\varepsilon_{\text{mach}} = (0.0001)_2$. Another related quantity is ULP, or unit in the last place.

- What does this say about the relative error incurred in floating point calculations?

  - The factor to get from one FP number to the next larger one is (mostly) independent of magnitude: $1 + \varepsilon_{\text{mach}}$.

  - Since we can't represent any results between

    $$x \quad \text{and} \quad x \cdot (1 + \varepsilon_{\text{mach}}),$$

    that's really the minimum error incurred.

  - In terms of relative error:

    $$\left|\frac{\tilde{x} - x}{x}\right| = \left|\frac{x(1 + \varepsilon_{\text{mach}}) - x}{x}\right| = \varepsilon_{\text{mach}}.$$

At least theoretically, $\varepsilon_{\text{mach}}$ is the maximum relative error in any FP operations. (Practical implementations do fall short of this.)

- What's that same number for double-precision floating point? (52 bits in the significand)

$$2^{-52} \approx 10^{-16}$$

We can expect FP math to consistently introduce relative errors of about 10^{-16}.

Working in double precision gives you about 16 (decimal) accurate digits.

Implementing Arithmetic

- How is floating point addition implemented?
  Consider adding $a = (1.101)_2 \cdot 2^1$ and $b = (1.001)_2 \cdot 2^{-1}$ in a system with three bits in the significand.

  Rough algorithm:

  1. Bring both numbers onto a common exponent
  2. Do grade-school addition from the front, until you run out of digits in your system.
  3. Round result.

$$
\begin{aligned}
a &= \ 1.\ 101 \cdot 2^1 \\
b &= \ 0.\ 01001 \cdot 2^1 \\
a + b &\approx \ 1.\ 111 \cdot 2^1
\end{aligned}
$$

Demo: Floating point and the harmonic series

Problems with FP Addition

- What happens if you subtract two numbers of very similar magnitude? As an example, consider $a = (1.1011)_2 \cdot 2^0$ and $b = (1.1010)_2 \cdot 2^0$.

$$\begin{aligned} a &= 1.\ 1011 \cdot 2^1 \\ b &= 1.\ 1010 \cdot 2^1 \\ a - b &\approx 0.\ 0001???? \cdot 2^1 \end{aligned}$$

or, once we normalize,

$$1.???? \cdot 2^{-3}.$$

There is no data to indicate what the missing digits should be.
→ Machine fills them with its 'best guess', which is not often good.

This phenomenon is called Catastrophic Cancellation.

Demo: Catastrophic Cancellation
In-class activity: Floating Point 2

Part 2:
Arrays–Computing with Many Numbers

6 Modeling the World with Arrays

6.1 The World in a Vector

Some Perspective

- We have so far (mostly) looked at what we can do with single numbers (and functions that return single numbers).
- Things can get *much* more interesting once we allow not just one, but *many* numbers together.
- It is natural to view an *array of numbers* as one object with its own rules.
  The simplest such set of rules is that of a vector.
- A 2D array of numbers can also be looked at as a matrix.
- So it's natural to use the tools of computational linear algebra.
- 'Vector' and 'matrix' are just *abstract structures* that come to life in many (*many!*) applications. The purpose of this section is to explore some of those applications.

Vectors

- What's a vector?

  A thing that defines *addition* and *scalar multiplication* with reasonable rules.

Vectors from a CS Perspective

- What would the concept of a vector look like in a programming language (e.g. Java)?

  In a sense, 'vector' is an *abstract interface*, like this:

  ```
  interface Vector {
    Vector add(Vector x, Vector y);
    Vector scale(Number alpha, Vector x);
  }
  ```

  (Along with guarantees that `add` and `multiply` interact sanely.)

Vectors in the ‘Real World’

Demo: Images as Vectors
Demo: Sound as Vectors
Demo: Shapes as Vectors

6.2 What can Matrices Do?

Matrices

What does a matrix do?

It represents a *linear function* between two vector spaces $f\colon U \to V$ in terms of bases $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ of U and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ of V. Let

$$\boldsymbol{u} = \alpha_1 \boldsymbol{u}_1 + \cdots + \alpha_n \boldsymbol{u}_n$$

and

$$\boldsymbol{v} = \beta_1 \boldsymbol{v}_1 + \cdots + \beta_m \boldsymbol{v}_m.$$

Then f can *always* be represented as a matrix that obtains the βs from the αs:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_m \end{pmatrix}.$$

Example: The 'Frequency Shift' Matrix

- Assume both $\boldsymbol{u}$ and $\mathbf{v}$ are linear combination of sounds of different frequencies:

$$\boldsymbol{u} = \alpha_1 \boldsymbol{u}_{110\ \mathrm{Hz}} + \alpha_2 \boldsymbol{u}_{220\ \mathrm{Hz}} + \cdots + \alpha_4 \boldsymbol{u}_{880\ \mathrm{Hz}}$$

(analogously for $\boldsymbol{v}$, but with βs). What matrix realizes a 'frequency doubling' of a signal represented this way?

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix}$$

Matrices in the 'Real World'

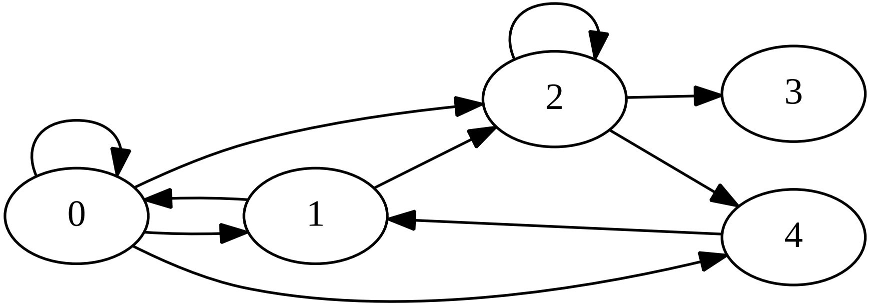
What are some examples of matrices in applications?

Demo: Matrices for Geometry Transformation
Demo: Matrices for Image Blurring
In-class activity: Computational Linear Algebra

6.3 Graphs

Graphs as Matrices

- How could this (directed) graph be written as a matrix?



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & & & & 1 \\ 1 & 1 & 1 & & \\ & & 1 & & \\ 1 & & 1 & & \end{pmatrix}$$

Matrices for Graph Traversal: Technicalities

- What is the general rule for turning a graph into a matrix?

  If there is an edge from node i to node j, then $A_{ji} = 1$.
  (otherwise zero)

- What does the matrix for an *undirected* graph look like?

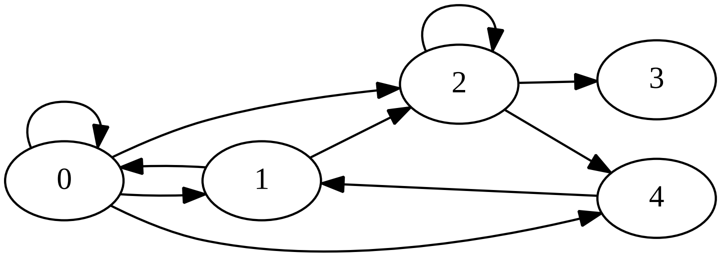
  Symmetric.

- How could we turn a *weighted graph* (i.e. one where the edges have weights–maybe 'pipe widths') into a matrix?

  Allow values other than zero and one for the entries of the matrix.

Graph Matrices and Matrix-Vector Multiplication

- If we multiply a graph matrix by the ith unit vector, what happens?



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & & & & 1 \\ 1 & 1 & 1 & & \\ & & 1 & & \\ 1 & & 1 & & \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

We get a vector that indicates (with a 1) all the nodes that are reachable from node i.

Demo: Matrices for Graph Traversal

[Next semester: Explicitly introduce normalized adj. matrices as Markov Chains]

6.4 Sparsity

Storing Sparse Matrices

- Some types of matrices (including graph matrices) contain many zeros.
  Storing all those zero entries is wasteful.
  How can we store them so that we avoid storing tons of zeros?

    - Python dictionaries (easy, but not efficient)
    - Using arrays...?

Storing Sparse Matrices Using Arrays

- How can we store a sparse matrix using just arrays? For example:

$$\begin{pmatrix} 0 & 2 & 0 & 3 \\ 1 & 4 & & \\ & & 5 & \\ 6 & & & 7 \end{pmatrix}$$

Idea: 'Compressed Sparse Row' ('CSR') format

- Write all non-zero *values* from top-left to bottom-right
- Write down what *column* each value was in
- Write down the index where each *row started*

$$\begin{aligned} \text{RowStarts} &= (\ 0\ \ 2\ \ 4\ \ 5\ \ 7\ ) \qquad \text{(zero-based)} \\ \text{Columns} &= (\ 1\ \ 3\ \ 0\ \ 1\ \ 2\ \ 0\ \ 3\ ) \qquad \text{(zero-based)} \\ \text{Values} &= (\ 2\ \ 3\ \ 1\ \ 4\ \ 5\ \ 6\ \ 7\ ) \end{aligned}$$

Demo: Sparse Matrices in CSR Format

7 Norms and Errors

Norms

- What's a norm?
  - A generalization of 'absolute value' to vectors.
  - $f(\boldsymbol{x})\colon \mathbb{R}^n \to \mathbb{R}_0^+$, returns a 'magnitude' of the input vector
  - In symbols: Often written $\|\boldsymbol{x}\|$.

- Define norm.

  A function $\|\boldsymbol{x}\|\colon \mathbb{R}^n \to \mathbb{R}_0^+$ is called a norm if and only if

  1. $\|\boldsymbol{x}\| > 0 \Leftrightarrow \boldsymbol{x} \neq \mathbf{0}$.
  2. $\|\gamma \boldsymbol{x}\| = |\gamma| \|\boldsymbol{x}\|$ for all scalars γ.
  3. Obeys triangle inequality $\|\boldsymbol{x} + \boldsymbol{y}\| \leqslant \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$

Examples of Norms

- What are some examples of norms?

  The so-called *p-norms*:

$$\left\| \begin{pmatrix} x_1 \\ \\ x_n \end{pmatrix} \right\|_p = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p} \quad (p \geqslant 1)$$

  $p = 1, 2, \infty$ particularly important

Demo: Vector norms

Norms and Errors

- If we're computing a vector result, the error is a vector.
  That's not a very useful answer to 'how big is the error'.
  What can we do?

  Apply a norm!

  How? Attempt 1:

  $$\text{Magnitude of error} \neq \|\text{true value}\| - \|\text{approximate value}\| \qquad \textbf{WRONG!}$$

  Attempt 2:

  $$\text{Magnitude of error} = \|\text{true value} - \text{approximate value}\|$$

Absolute and Relative Error

- What are the absolute and relative errors in approximating the location of Siebel center $(40.114, -88.224)$ as $(40, -88)$ using the 2-norm?

$$\begin{pmatrix} 40.114 \\ -88.224 \end{pmatrix} - \begin{pmatrix} 40 \\ -88 \end{pmatrix} = \begin{pmatrix} 0.114 \\ -.224 \end{pmatrix}$$

Absolute magnitude;

$$\left\| \begin{pmatrix} 40.114 \\ -88.224 \end{pmatrix} \right\|_2 \approx 96.91$$

Absolute error:

$$\left\| \begin{pmatrix} 0.114 \\ -.224 \end{pmatrix} \right\|_2 \approx .2513$$

Relative error:

$$\frac{.2513}{96.91} \approx .00259.$$

But: Is the 2-norm really the right norm here?

Demo: Calculate geographic distances using `tripstance.com`

Matrix Norms

- What norms would we apply to matrices?

  - Easy answer: '*Flatten*' matrix as vector, use vector norm.
    Bad, not very meaningful.
    Still, the so-called Frobenius norm

    $$\|A\|_F := \sqrt{\sum_{i,j} a_{ij}^2}$$

    does exactly that. (*But:* This is not actually a matrix norm in the sense below.)

  - Instead: Choose norms for matrices to interact meaningfully with an 'associated' vector norm $\|\cdot\|$ so that $\|A\|$ obeys

    $$\|A\boldsymbol{x}\| \leqslant \|A\|\|\boldsymbol{x}\|.$$

- This can be achieved by choosing, for a given vector norm $\|\cdot\|$,

$$\|A\| := \max_{\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|.$$

  This is called the matrix norm associated with the vector norm $\|\cdot\|$.

- The following is equivalent:

$$\max_{\|\boldsymbol{x}\|\neq\boldsymbol{0}} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{\|\boldsymbol{x}\|\neq\boldsymbol{0}} \left\| A \underbrace{\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}}_{\boldsymbol{y}} \right\| \overset{\|\boldsymbol{y}\|=1}{=} \max_{\|\boldsymbol{y}\|=1} \|A\boldsymbol{y}\| = \|A\|.$$

- Logically, for each vector norm, we get a different matrix norm, so that, e.g. for the vector 2-norm $\|\boldsymbol{x}\|_2$ we get a matrix 2-norm $\|A\|_2$, and for the vector ∞-norm $\|\boldsymbol{x}\|_\infty$ we get a matrix ∞-norm $\|A\|_\infty$.

Demo: Matrix norms

In-class activity: Matrix norms

Properties of Matrix Norms

Matrix norms inherit the vector norm properties:

1. $\|A\| > 0 \Leftrightarrow A \neq \mathbf{0}$.
2. $\|\gamma A\| = |\gamma| \|A\|$ for all scalars γ.
3. Obeys triangle inequality $\|A + B\| \leqslant \|A\| + \|B\|$

- But also some more properties that stem from our definition:
  1. $\|A\boldsymbol{x}\| \leqslant \|A\| \|\boldsymbol{x}\|$
  2. $\|A\,B\| \leqslant \|A\| \|B\|$ (easy consequence)

  Both of these are called submultiplicativity of the matrix norm.

Example: Orthogonal Matrices

- What is the 2-norm of an orthogonal matrix?

  Linear Algebra recap: For an orthogonal matrix A, $A^{-1} = A^T$.

  In other words: $A\,A^T = A^T A = I$.
  Next:

$$\|A\|_2 = \max_{\|\boldsymbol{x}\|_2=1} \|A\boldsymbol{x}\|_2$$

  where

$$\|A\boldsymbol{x}\|_2 = \sqrt{(A\boldsymbol{x})^T(A\boldsymbol{x})} = \sqrt{\boldsymbol{x}^T A^T A\boldsymbol{x}} = \sqrt{\boldsymbol{x}^T(A^T A)\boldsymbol{x}} = \sqrt{\boldsymbol{x}^T\boldsymbol{x}} = \|\boldsymbol{x}\|_2,$$

  so $\|A\|_2 = 1$.

Conditioning

- Now, let's study condition number of solving a linear system

$$A\boldsymbol{x} = \boldsymbol{b}.$$

Input: $\boldsymbol{b}$ with error $\Delta \boldsymbol{b}$,

Output: $\boldsymbol{x}$ with error $\Delta \boldsymbol{x}$.

Observe $A(\boldsymbol{x} + \Delta \boldsymbol{x}) = (\boldsymbol{b} + \Delta \boldsymbol{b})$, so $A\Delta \boldsymbol{x} = \Delta \boldsymbol{b}$.

$$\begin{aligned}
\frac{\text{rel err. in output}}{\text{rel err. in input}} &= \frac{\|\Delta \boldsymbol{x}\| / \|\boldsymbol{x}\|}{\|\Delta \boldsymbol{b}\| / \|\boldsymbol{b}\|} = \frac{\|\Delta \boldsymbol{x}\| \|\boldsymbol{b}\|}{\|\Delta \boldsymbol{b}\| \|\boldsymbol{x}\|} \\
&= \frac{\|A^{-1}\Delta \boldsymbol{b}\| \|A\boldsymbol{x}\|}{\|\Delta \boldsymbol{b}\| \|\boldsymbol{x}\|} \\
&\leqslant \|A^{-1}\| \|A\| \frac{\|\Delta \boldsymbol{b}\| \|\boldsymbol{x}\|}{\|\Delta \boldsymbol{b}\| \|\boldsymbol{x}\|} \\
&= \|A^{-1}\| \|A\|.
\end{aligned}$$

So we've found an *upper bound* on the condition number. With a little bit of fiddling, it's not too hard to find examples that achieve this bound, i.e. that it is *sharp*.

So we've found the condition number of linear system solving, also called the condition number of the matrix A:

$$\text{cond}(A) = \kappa(A) = \|A\| \|A^{-1}\|.$$

- cond is relative to a given norm. So, to be precise, use

$$\text{cond}_2 \quad \text{or} \quad \text{cond}_\infty.$$

- If A^{-1} does not exist: $\text{cond}(A) = \infty$ by convention.

Demo: Condition number visualized
Demo: Conditioning of 2×2 Matrices

More Properties of the Condition Number

- What is $\text{cond}(A^{-1})$?

  $\text{cond}(A^{-1}) = \|A\| \cdot \|A^{-1}\| = \text{cond}(A)$.

- What is the condition number of applying the matrix-vector multiplication $A\boldsymbol{x} = \boldsymbol{b}$? (I.e. now $\boldsymbol{x}$ is the input and $\boldsymbol{b}$ is the output)

  Let $B = A^{-1}$.
  Then computing $\boldsymbol{b} = A\boldsymbol{x}$ is equivalent to *solving* $B\boldsymbol{b} = \boldsymbol{x}$.
  Solving $B\boldsymbol{b} = \boldsymbol{x}$ has condition number $\text{cond}(B) = \text{cond}(A^{-1}) = \text{cond}(A)$.

  So the operation 'multiply a vector by matrix A' has the same condition number as 'solve a linear system with matrix A'.

Matrices with Great Conditioning (Part 1)

- Give an example of a matrix that is *very* well-conditioned.
  (I.e. has a condition-number that's *good* for computation.)
  What is the best possible condition number of a matrix?

  Small condition numbers mean *not a lot of error amplification*.
  Small condition numbers are good.

  The identity matrix I should be well-conditioned:

  $$\|I\| = \max_{\|\boldsymbol{x}\|=1} \|I\boldsymbol{x}\| = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{x}\| = 1.$$

  It turns out that this is the smallest possible condition number:

  $$1 = \|I\| = \|A \cdot A^{-1}\| \leqslant \|A\| \cdot \|A^{-1}\| = \kappa(A).$$

  Both of these are true for any norm $\|\cdot\|$.

Matrices with Great Conditioning (Part 2)

- What is the 2-norm condition number of an orthogonal matrix A?

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \|A\|_2\|A^T\|_2 = 1.$$

  That means *orthogonal matrices have optimal conditioning*.
  They're very well-behaved in computation.

In-class activity: Matrix Conditioning

8 The ‘Undo’ Button for Linear Operations: LU

Solving Systems

- Want methods/algorithms to solve linear systems. Starting small, a kind of system that's easy to solve has a ... matrix.

  'Triangular'. $\rightarrow$ Easy to solve by hand.
  Upper/lower triangular matrices.

Triangular matrices

- Solve

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}.$$

  - Rewrite as individual equations.
  - This process is called back-substitution.
  - The analogous process for lower triangular matrices is called forward substitution.

Demo: Coding back-substitution

More General Matrices

- What about non-triangular matrices?

  Can do Gaussian Elimination, just like in linear algebra class.

Gaussian Elimination

Demo: Vanilla Gaussian Elimination

- What do we get by doing Gaussian Elimination?
  Row Echelon Form.

- How is that different from being upper triangular?
  Zeros allowed on and above the diagonal.

- What if we do not just eliminate downward but also upward?
  That's called Gauss-Jordan elimination. Turns out to be computationally inefficient. We won't look at it.

Elimination Matrices

- What does this matrix do?

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}$$

  - Add $(-1/2)\times$ the first row to the third row.
  - One elementary step in Gaussian elimination
  - Matrices like this are called Elimination Matrices

About Elimination Matrices

- Are elimination matrices invertible?

  Sure! Inverse of

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ -\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}$$

  should be

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ +\frac{1}{2} & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}.$$

More on Elimination Matrices

Demo: Elimination matrices I

- **Idea:** With enough elimination matrices, we should be able to get a matrix into row echelon form.

$$M_\ell M_{\ell-1} \cdots M_2 M_1 A = \langle \text{Row Echelon Form } U \text{ of } A \rangle.$$

- So what do we get from many combined elimination matrices like that? (a lower triangular matrix)

Demo: Elimination Matrices II

Summary on Elimination Matrices

- El.matrices with off-diagonal entries in a single column just “merge” when multiplied by one another.
- El.matrices with off-diagonal entries in different columns merge when we multiply (left-column) * (right-column) but not the other way around.
- Inverse: Flip sign below diagonal

LU Factorization

- Can build a factorization from elimination matrices. How?

$$A = \underbrace{M_1^{-1} M_2^{-1} \overset{-1}{\cdots} M_{\ell-1}^{-1} M_\ell^{-1}}_{\text{lower } \triangle \text{ mat } \boldsymbol{L}} U = L\,U.$$

  This is called LU factorization (or LU decomposition).

- Does this help solve $A\boldsymbol{x} = \boldsymbol{b}$?

$$\begin{aligned} A\boldsymbol{x} &= \boldsymbol{b} \\ L\underbrace{U\boldsymbol{x}}_{\boldsymbol{y}} &= \boldsymbol{b} \\ L\boldsymbol{y} &= \boldsymbol{b} \quad \leftarrow \quad \text{solvable by fwd. subst.} \\ U\boldsymbol{x} &= \boldsymbol{y} \quad \leftarrow \quad \text{solvable by bwd. subst.} \end{aligned}$$

  Now know $\boldsymbol{x}$ that solves $A\boldsymbol{x} = \boldsymbol{b}$.

Demo: LU factorization

In-class activity: LU Factorization

LU: Failure Cases?

- Is LU/Gaussian Elimination bulletproof?
  No, very much not:

$$A = \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix}.$$

Q: Is this a problem with the process or with the entire idea of LU?

$$\begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & \\ \ell_{21} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ & u_{22} \end{pmatrix}$$

So

$$\begin{matrix} & \begin{pmatrix} u_{11} & u_{12} \\ & u_{22} \end{pmatrix} \\ \begin{pmatrix} 1 & \\ \ell_{21} & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix} \end{matrix} \quad \to \quad u_{11} = 0$$

$$\underbrace{u_{11} \cdot \ell_{21}}_{0} + 1 \cdot 0 = 2$$

It turns out to be that A doesn't *have* an LU factorization.

- What can be done to get something *like* an LU factorization?

  Idea: In Gaussian elimination: simply swap rows, equivalent linear system.

  Approach:

  - Good Idea: Swap rows if there's a zero in the way
  - Even better Idea: Find the largest entry (by absolute value), swap it to the top row.

  The entry we divide by is called the pivot.
  Swapping rows to get a bigger pivot is called (partial) pivoting.

Fixing nonexistence of LU

- How do we capture 'row switches' in a factorization?

$$\underbrace{\begin{pmatrix} 1 & & & \\ & & 1 & \\ & 1 & & \\ & & & 1 \end{pmatrix}}_{P}\begin{pmatrix} A & A & A & A \\ B & B & B & B \\ C & C & C & C \\ D & D & D & D \end{pmatrix} = \begin{pmatrix} A & A & A & A \\ \color{red}{C} & \color{red}{C} & \color{red}{C} & \color{red}{C} \\ \color{red}{B} & \color{red}{B} & \color{red}{B} & \color{red}{B} \\ D & D & D & D \end{pmatrix}.$$

P is called a permutation matrix.

Q: What's P^{-1}?

- What does this process look like then?

$$
\begin{aligned}
P_1A &\quad \text{Pivot first column}\\
M_1P_1A &\quad \text{Eliminate first column}\\
P_2M_1P_1A &\quad \text{Pivot second column}\\
M_2P_2M_1P_1A &\quad \text{Eliminate second column}\\
P_3M_2P_2M_1P_1A &\quad \text{Pivot third column}\\
M_3P_3M_2P_2M_1P_1A &\quad \text{Eliminate third column}
\end{aligned}
$$

Or

$$A = P_1M_1^{-1}P_2M_2^{-1}P_3M_3^{-1}U.$$

With some more thought, it is possible to arrive at

$$P\,A = L\,U,$$

where P is a permutation matrix.

Computational Cost

- What is the computational cost of multiplying two $n \times n$ matrices?

  $O(n^3)$

- What is the computational cost of carrying out LU factorization on an $n \times n$ matrix?

  Recall

  $$M_3 M_2 M_1 A = U \ldots$$

  so $O(n^4)$?!!!

  Fortunately not: Multiplications with permutation matrices and elimination matrices only cost $O(n^2)$.

So overall cost of LU is just $O(n^3)$.

Demo: Complexity of Mat-Mat multiplication and LU

More cost concerns

- What's the cost of solving $A\boldsymbol{x} = \boldsymbol{b}$?
  LU: $O(n^3)$
  FW/BW Subst: $2 \times O(n^2) = O(n^2)$

- What's the cost of solving $A\boldsymbol{x} = \boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n$?
  LU: $O(n^3)$
  FW/BW Subst: $2n \times O(n^2) = O(n^3)$

- What's the cost of finding A^{-1}?
  Same as solving

$$AX = I,$$

  so still $O(n^3)$.

Cost: Worrying about the Constant, BLAS

$O(n^3)$ really means

$$\alpha \cdot n^3 + \beta \cdot n^2 + \gamma \cdot n + \delta.$$

All the non-leading and constants terms swept under the rug. But: at least the leading constant ultimately matters.

Getting that constant to be small is surprisingly hard, even for something deceptively simple such as matrix-matrix multiplication.

Idea: Rely on library implementation: BLAS (Fortran)

Level 1	$\boldsymbol{z} = \alpha \boldsymbol{x} + \boldsymbol{y}$	vector-vector operations $O(n)$ `?axpy`
Level 2	$\boldsymbol{z} = A\boldsymbol{x} + \boldsymbol{y}$	matrix-vector operations $O(n^2)$ `?gemv`
Level 3	$C = A\,B + \beta C$	matrix-matrix operations $O(n^3)$ `?gemm`

LAPACK: Implements 'higher-end' things (such as LU) using BLAS
Special matrix formats can also help save const significantly, e.g.

- banded
- sparse

LU: Special cases

- What happens if we feed a non-invertible matrix to LU?

  $$PA = LU$$

  (invertible, not invertible) (Why?)

- What happens if we feed LU an $m \times n$ non-square matrices?

  Think carefully about sizes of factors and columns/rows that do/don't matter. Two cases:

  - $m > n$ (tall&skinny): L: $m \times n$, U: $n \times n$
  - $m < n$ (short&fat): L: $m \times m$, U: $m \times n$

  This is called reduced LU factorization.

In-class activity: LU factorization 2

9 LU: Applications

9.1 Linear Algebra Applications

Solve a Linear System

- LU factorization gives us $PA = LU$, so that P is a permutation matrix, L is lower triangular, U is upper triangular. How does that help solve a linear system $A\boldsymbol{x} = \boldsymbol{b}$?

$$\begin{aligned} A\boldsymbol{x} &= \boldsymbol{b} \\ \Leftrightarrow PA\boldsymbol{x} &= P\boldsymbol{b} \\ \Leftrightarrow L\underbrace{U\boldsymbol{x}}_{\text{Call this } \boldsymbol{y}} &= P\boldsymbol{b} \end{aligned}$$

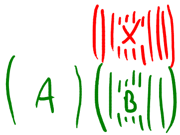
1. Then solve $L\boldsymbol{y} = P\boldsymbol{b}$ using forward substitution.
2. Then solve $U\boldsymbol{x} = \boldsymbol{y}$ using backward substitution.

Q: Isn't this complicated or expensive?
(No: The factorization itself is cheap at least compared to the other $O(n^3)$ things one might do–and reusable.)

Solve a Matrix Equation

- Suppose we want to solve $AX = B$.
  A and B are given, X is unknown.
  (Assume: square and have same size) How can we do that using LU?

  Column-by-column:



  - No different than solving lots of linear systems with the same A and lots of different right-hand side vectors b.
  - Can reuse P, L, and U for every column.

Compute an Inverse

- Suppose we want to compute the inverse A^{-1} of a matrix A.
  How do we do that using LU?

  The inverse solves the matrix equation $AX = I$.
  (where I is the identity matrix)

  $\rightarrow$ Can find inverse A^{-1} using LU, column-by-column.

- What's the computational cost of doing so?

$$\underbrace{O(n^3)}_{\text{LU}} + n \cdot \underbrace{O(n^2)}_{\text{FW/BW subst}} = O\left(n^3\right)$$

Find the Determinant of a Matrix

- How can we find the determinant of a matrix using LU?

$$
\begin{aligned}
PA &= LU \\
\underbrace{\det(P)}_{\pm 1}\det(A) &= \underbrace{\det(L)}_{1}\underbrace{\det(U)}_{\text{Product of diagonal}}
\end{aligned}
$$

Find Row Echelon Form... if we can?

- The factor U in pivoted LU looks like it is in upper echelon form. Is it?

  Turns out *no*, it is not. For example, U can contain linearly dependent rows.

  Demo: LU and Upper Echelon Form

  Echelon form would allow us to determine (e.g.) the rank (=the number of linearly independent rows/columns) of a matrix A. (or the nullspace, or a basis, or...) Is doing this actually a good idea?

Finding the Rank of a Matrix Numerically... if we can?

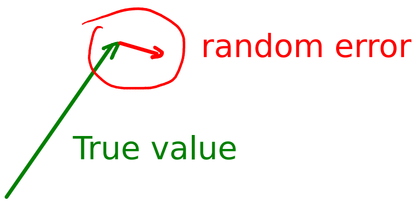
- Can we find the rank of a matrix numerically?

  The simplest version of that task is to test whether (just) two vectors are linearly dependent, i.e. point in the same direction, i.e. are multiples of each other: Is there an α so that

  $$\boldsymbol{u} = \alpha \boldsymbol{v}?$$

  Every floating point operation adds small–assume 'random'–error, i.e.

  $$\boldsymbol{u}_{\text{num}} = \boldsymbol{u}_{\text{true}} + \Delta \boldsymbol{u}.$$

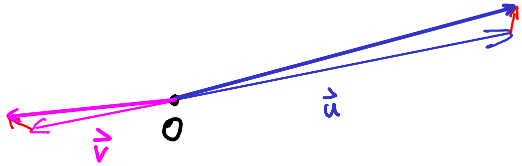


  Next, observe:

  - Two randomly vectors almost surely do no not point in the same direction.

- Two random vectors are almost surely *not* linearly dependent.

Suppose we would like to test two inexact vectors for linear dependence:



- True: $\boldsymbol{u} = \alpha \boldsymbol{v}$ (linearly dependent)
- Computed: $\boldsymbol{u} \neq \alpha \boldsymbol{v}$ (not linearly dependent)

Therefore:

- Asking for 'rank' or 'linear dependence' without specifying a tolerance makes no sense.
- LU does not allow specifying a tolerance $\rightarrow$ wrong tool for the job.

9.2 Interpolation

Recap: Interpolation

Starting point: Looking for a linear combination of functions φ_i to hit given data points (x_i, y_i).

Interpolation becomes solving the linear system:

$$y_i = f(x_i) = \sum_{j=0}^{N_{\text{func}}} \alpha_j \underbrace{\varphi_j(x_i)}_{V_{ij}} \qquad \leftrightarrow \qquad V\boldsymbol{\alpha} = \boldsymbol{y}.$$

Want unique answer: Pick $N_{\text{func}} = N \to V$ square.

V is called the (generalized) Vandermonde matrix.

Main lesson:

$$V(\text{coefficients}) = (\text{values at nodes}).$$

Rethinking Interpolation

We have so far always used monomials $(1, x, x^2, x^3, \ldots)$ and equispaced points for interpolation. It turns out that this has *significant problems*.

Demo: Monomial interpolation

Demo: Choice of Nodes for Polynomial Interpolation

Interpolation: Choosing Basis Function and Nodes

Both function basis and point set are under our control. What do we pick?

Ideas for basis functions:

- Monomials $1, x, x^2, x^3, x^4, \ldots$
- Functions that make $V = I$ $\rightarrow$ 'Lagrange basis'
- Functions that make V triangular $\rightarrow$ 'Newton basis'
- Splines (piecewise polynomials)
- Orthogonal polynomials
- Sines and cosines
- 'Bumps' ('Radial Basis Functions')

Ideas for nodes:

- Equispaced
- 'Edge-Clustered' (so-called Chebyshev/Gauss/... nodes)

Better Conditioning: Orthogonal Polynomials

- What caused monomials to have a terribly conditioned Vandermonde?
  Being close to linearly dependent.

- What's a way to make sure two vectors are *not* like that?
  Orthogonality

- But polynomials are functions!
  How can those be orthogonal? Just need something like a dot product!

$$\boldsymbol{f} \cdot \boldsymbol{g} = \sum_{i=1}^{n} f_i g_i = \langle \boldsymbol{f}, \boldsymbol{g} \rangle$$

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x) \mathrm{d}x$$

  Orthogonal then just means $\langle f, g \rangle = 0$.
  Q: How can we find an orthogonal basis?
  A: Apply Gram-Schmidt to the monomials.

Obtained Legendre polynomials.
Demo: Orthogonal polynomials

- But how can I practically compute the Legendre polynomials?
  $\rightarrow$ DLMF, Chapter on orthogonal polynomials
  Main lessons:
  - There exist three-term recurrences. Easy to apply if you know the first two.
  - There is a whole zoo of polynomials there, depending on the weight function w in the (generalized) inner product:

    $$\langle f, g \rangle = \int w(x) f(x) g(x) \mathrm{d}x.$$

    Some sets of orthogonal polynomials live on intervals other than $(-1, 1)$.

Another Family of Orthogonal Polynomials: Chebyshev

Three equivalent definitions:

- Result of Gram-Schmidt with weight $1/\sqrt{1-x^2}$
  - What is that weight?
    $1/$(Half circle), i.e. $x^2+y^2=1$, with $y=\sqrt{1-x^2}$
- $T_k(x)=\cos(k\cos^{-1}(x))$
- $T_k(x)=2x\,T_k(x)-T_{k-1}(x)$

Demo: Chebyshev interpolation part I

- What are good nodes to use with Chebyshev polynomials?

  The answer would be particularly simple if the nodes were $\cos(*)$.
  So why not cos(equispaced)?

  Might get

$$x_i=\cos\left(\frac{i}{k}\pi\right)\qquad(i=0,1,\ldots,k)$$

Chebyshev Nodes

Might also consider zeros (instead of roots) of T_k:

$$x_i = \cos\left(\frac{2i+1}{2k}\pi\right) \quad (i = 1\ldots, k).$$

The Vandermonde for these (with T_k) can be applied in $O(N \log N)$ time, too.

It turns out that we were still looking for a good set of interpolation nodes.

- We came up with the criterion that the nodes should bunch towards the ends. Do these do that?
  Yes.

Demo: Chebyshev interpolation part II

Calculus on Interpolants

- Suppose we have an interpolant $\tilde{f}(x)$ with $f(x_i) = \tilde{f}(x_i)$ for $i = 1, \ldots, n$:

  $$\tilde{f}(x) = \alpha_1 \varphi_1(x) + \cdots + \alpha_n \varphi_n(x)$$

  How do we compute the derivative of $\tilde{f}$?

  $$\tilde{f}'(x) = \alpha_1 \varphi_1'(x) + \cdots + \alpha_n \varphi_n'(x).$$

  Easy because interpolation basis (φ_i) is known.

- Suppose we have function values at nodes $(x_i, f(x_i))$ for $i = 1, \ldots, n$ for a function f. If we want $f'(x_i)$, what can we do?

  $f'(x_i)$: Hard to get
  $\tilde{f}'(x_i)$: Easy to get

  So:

  1. Compute coefficients $\boldsymbol{\alpha} = V^{-1}\boldsymbol{f}$, where $\boldsymbol{f} = (f(x_1), \ldots, f(x_n))^T$.

2. Build generalized Vandermonde with *derivatives* of basis:

$$V' = \begin{pmatrix} \varphi_1'(x_1) & \cdots & \varphi_n'(x_1) \\ \vdots & & \vdots \\ \varphi_1'(x_n) & \cdots & \varphi_n'(x_n) \end{pmatrix}.$$

3. Compute

$$V'\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1\varphi_1'(x_1) + \cdots \alpha_n\varphi_n'(x_1) \\ \\ \\ \end{pmatrix} = \underbrace{\begin{pmatrix} \tilde{f}'(x_1) \\ \vdots \\ \tilde{f}'(x_n) \end{pmatrix}}_{\tilde{\boldsymbol{f}}'}.$$

All in one step: $\tilde{\boldsymbol{f}}' = V'V^{-1}\boldsymbol{f}$.
In other words: $V'V^{-1}$ is a matrix to apply a derivative!
We call $D = V'V^{-1}$ a differentiation matrix.

About Differentiation Matrices

- How could you find coefficients of the derivative?

  $\boldsymbol{\alpha}' = V^{-1} V' V^{-1} \boldsymbol{f}.$

- Give a matrix that finds the second derivative.

  $V' V^{-1} V' V^{-1}.$

Demo: Taking derivatives with Vandermonde matrices

Finite Difference Formulas

- It is possible to use the process above to find 'canned' formulas for taking derivatives. Suppose we use three points equispaced points ($x-h$, x, $x+h$) for interpolation (i.e. a degree-2 polynomial).
  - What is the resulting differentiation matrix?
  - What does it tell us?

$$D = V'V^{-1} = \begin{pmatrix} \cdots & \cdots & \cdots \\ -1/2h & 0 & 1/2h \\ \cdots & \cdots & \cdots \end{pmatrix}$$

(Can find the dependence on h by varying h and watching the entries.) When we apply that, we get

$$V'V^{-1} \begin{pmatrix} f(x-h) \\ f(x) \\ f(x+h) \end{pmatrix} = \begin{pmatrix} \cdots \\ \frac{f(x+h)-f(x-h)}{2h} \\ \cdots \end{pmatrix}$$

So we can compute an approximate (second-order accurate!) derivative just by using this formula.

Generalizes to more (and non-center) points easily.

Computing Integrals with Interpolation

- Can we use a similar process to compute (approximate) integrals of a function f?

  The process of computing approximate integrals is called 'quadrature'.
  Same idea as derivatives: interpolate, then integrate.

  Have: interpolant $\tilde{f}(x) = \alpha_1\varphi_1(x) + \cdots + \alpha_n\varphi_n(x)$
  so that $\tilde{f}(x_i) = f(x_i) = y_i$. We'll call the x_i the quadrature nodes.

  Want: Integral

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &\approx \int_a^b \tilde{f}(x)\mathrm{d}x = \int_a^b \alpha_1\varphi_1(x) + \cdots + \alpha_n\varphi_n(x)\mathrm{d}x \\ &= \alpha_1\int_a^b \varphi_1(x)\mathrm{d}x + \cdots + \alpha_n\int_a^b \varphi_n(x)\mathrm{d}x.\end{aligned}$$

Idea: $d_i = \int_a^b \varphi_i(x)\mathrm{d}x$ can be computed ahead of time, so that

$$\int_a^b \tilde{f}(x)\mathrm{d}x = \alpha_1 d_1 + \cdots + \alpha_n d_n = \boldsymbol{d}^T \boldsymbol{\alpha} = \boldsymbol{d}^T (V^{-1}\boldsymbol{y}) = (\boldsymbol{d}^T V^{-1})\boldsymbol{y}.$$

Can call $\boldsymbol{w} := V^{-T}\boldsymbol{d}$ the quadrature weights and compute

$$\int_a^b \tilde{f}(x)\mathrm{d}x = \boldsymbol{w}^T\boldsymbol{y} = \boldsymbol{w}\cdot\boldsymbol{y}.$$

Example: Building a Quadrature Rule

Demo: Computing the Weights in Simpson's Rule

- Suppose we know

$$f(x_0) = 2 \qquad f(x_1) = 0 \qquad f(x_2) = 3$$

$$x_0 = 1 \qquad x_1 = \frac{1}{2} \qquad x_2 = 1$$

How can we find an approximate integral?

1. Find coefficients

$$\boldsymbol{\alpha} = V^{-1}\begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}.$$

2. Compute integrals

$$\begin{aligned}\int_0^1 1\,\mathrm{d}x &= 1\\ \int_0^1 x\,\mathrm{d}x &= \frac{1}{2}\\ \int_0^1 x^2\,\mathrm{d}x &= \left[\frac{1}{3}x^3\right]_0^1 = \frac{1}{3}\end{aligned}$$

3. Combine it all together:

$$\int_0^1 \tilde{f}(x)\mathrm{d}x = \underbrace{\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \end{pmatrix} V^{-1}}_{\text{weights } \boldsymbol{w}} \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} .167 \\ .667 \\ .167 \end{pmatrix} \cdot \begin{pmatrix} f(0) \\ f(1/2) \\ f(1) \end{pmatrix}.$$

It turns out that this rule has someone's name attached to it. It's called Simpson's rule.

Facts about Quadrature

- What does Simpson's rule look like on $[0, 1/2]$?

$$\frac{1}{2}\begin{pmatrix} .167 \\ .667 \\ .167 \end{pmatrix} \cdot \begin{pmatrix} f(0) \\ f(1/2) \\ f(1) \end{pmatrix}$$

- What does Simpson's rule look like on $[5, 6]$?

$$\begin{pmatrix} .167 \\ .667 \\ .167 \end{pmatrix} \cdot \begin{pmatrix} f(5) \\ f(5.5) \\ f(6) \end{pmatrix}$$

- How accurate is Simpson's rule?

Demo: Accuracy of Simpson's rule

- Quadrature:

$$\left|\int_a^b f(x)\mathrm{d}x - \int_a^b \tilde{f}(x)\mathrm{d}x\right| \leqslant C \cdot h^{n+2}$$

  (where $h = b - a$)
  (Due to a happy accident, odd n produce an even smaller error.)

- Interpolation:

$$\max_{x\in[a,b]} |f(x) - \tilde{f}(x)| \leqslant C \cdot h^{n+1}$$

- Differentiation:

$$\max_{x\in[a,b]} \left|f'(x) - \tilde{f}'(x)\right| \leqslant C \cdot h^{n}$$

General lesson: More derivatives $\Rightarrow$ Worse accuracy.

10 Repeating Linear Operations: Eigenvalues and Steady States

Eigenvalue Problems: Setup/Math Recap

A is an $n \times n$ matrix.

- $\boldsymbol{x} \neq \mathbf{0}$ is called an eigenvector of A if there exists a λ so that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

- In that case, λ is called an eigenvalue.

Finding Eigenvalues

- How do you find eigenvalues?

  Linear Algebra approach:

$$
\begin{aligned}
& A\boldsymbol{x} = \lambda \boldsymbol{x} \\
\Leftrightarrow\ & (A - \lambda I)\boldsymbol{x} = 0 \\
\Leftrightarrow\ & A - \lambda I \text{ singular} \\
\Leftrightarrow\ & \det(A - \lambda I) = 0
\end{aligned}
$$

  $\det(A - \lambda I)$ is called the characteristic polynomial, which has degree n, and therefore n (potentially complex) roots.

  Q: Does that help computationally?
  A: Abel showed that for $n \geqslant 5$ there is no general formula for the roots of the polynomial. (i.e. no analog to the quadratic formula for $n = 5$)

Algorithmically, that means we will need to *approximate*. So far (e.g. for LU and QR), if it had not been for FP error, we would have obtained *exact* answers. For eigenvalue problems, that is no longer true–we can only hope for an *approximate* answer.

Transforming Eigenvalue Problems

Suppose we know that $A\boldsymbol{x} = \lambda\boldsymbol{x}$. What are the eigenvalues of these changed matrices?

- Shift. $A \to A - \sigma I$

$$(A - \sigma I)\boldsymbol{x} = (\lambda - \sigma)\boldsymbol{x}$$

- Inversion. $A \to A^{-1}$

$$A^{-1}\boldsymbol{x} = \lambda^{-1}\boldsymbol{x}$$

- Power. $A \to A^k$

$$A^k\boldsymbol{x} = \lambda^k\boldsymbol{x}$$

- Inverse. $A \to A^{-1}$

$$A^{-1}\boldsymbol{x} = \lambda^{-1}\boldsymbol{x}$$

Changing Eigenvectors

- Suppose $A\boldsymbol{x} = \lambda\boldsymbol{x}$.
  Can we change the eigenvectors? (but leave the eigenvalues the same)

  Consider a so-called similarity transform with an invertible matrix T:

  $$T^{-1}A\,T.$$

  Q: Can we find an eigenvector of that matrix?
  A: Let $\boldsymbol{y} := T^{-1}\boldsymbol{x}$. Then

  $$T^{-1}A\,T\boldsymbol{y} = \lambda\boldsymbol{y}$$

  Matrices A and B are called similar if there is a T so that $B = T^{-1}A\,T$.

Diagonalizability

- When is a matrix called diagonalizable?

  Assume a matrix A has a n linear independent eigenvectors (i.e. a full basis of them). Call those $(\boldsymbol{x}_n)_{i=1}^n$.

  In that case, let

$$X = \begin{pmatrix} | & & | \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \\ | & & | \end{pmatrix},$$

  and observe

$$\begin{aligned} AX &= XD \\ \Leftrightarrow \quad A &= XDX^{-1}, \end{aligned}$$

  where D is a diagonal matrix with the eigenvalues.

  In that sense: "Diagonalizable" = "Similar to a diagonal matrix".

Are all Matrices Diagonalizable?

- Give characteristic polynomial, eigenvalues, eigenvectors of

$$\begin{pmatrix} 1 & 1 \\ & 1 \end{pmatrix}.$$

CP: $(\lambda - 1)^2$
Eigenvalues: 1 (with multiplicity 2)
Eigenvectors:

$$\begin{pmatrix} 1 & 1 \\ & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$\Rightarrow x + y = x \Rightarrow y = 0$. So all eigenvectors must look like $\begin{pmatrix} x \\ 0 \end{pmatrix}$. Eigenvector matrix X won't be invertible. $\rightarrow$ This matrix is *not diagonalizable*!

Power Iteration

- What are the eigenvalues of A^{1000}?

  Assume $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ with eigenvectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.
  Further assume $\|\boldsymbol{x}_i\| = 1$.

  Use $\boldsymbol{x} = \alpha \boldsymbol{x}_1 + \beta \boldsymbol{x}_2$.

$$\boldsymbol{y} = A^{1000}(\alpha \boldsymbol{x}_1 + \beta \boldsymbol{x}_2) = \alpha \lambda_1^{1000} \boldsymbol{x}_1 + \beta \lambda_2^{1000} \boldsymbol{x}_2$$

  Or

$$\frac{\boldsymbol{y}}{\lambda_1^{1000}} = \alpha \boldsymbol{x}_1 + \beta \underbrace{\left(\underbrace{\frac{\lambda_2}{\lambda_1}}_{<1} \right)^{1000}}_{\ll 1} \boldsymbol{x}_2.$$

  Idea: Use this as a computational procedure to find $\boldsymbol{x}_1$.
  Called Power Iteration.

Power Iteration: Issues?

- What could go wrong with Power Iteration?
  - Starting vector has no component along $\boldsymbol{x}_1$
    Not a problem in practice: Rounding will introduce one.
  - Overflow in computing λ_1^{1000}
    $\to$ Normalized Power Iteration
  - $\lambda_1 = \lambda_2$
    Real problem.

What about Eigenvalues?

- Power Iteration generates eigenvectors. What if we would like to know eigenvalues?

  Estimate them:

  $$\frac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

  - $=\lambda$ if $\boldsymbol{x}$ is an eigenvector w/ eigenvalue λ
  - Otherwise, an estimate of a ‘nearby’ eigenvalue

  This is called the Rayleigh quotient.

Convergence of Power Iteration

- What can you say about the convergence of the power method? Say $\boldsymbol{v}_1^{(k)}$ is the kth estimate of the eigenvector $\boldsymbol{x}_1$, and

  $$e_k = \left\| \boldsymbol{x}_1 - \boldsymbol{v}_1^{(k)} \right\|.$$

  Easy to see:

  $$e_{k+1} \approx \frac{|\lambda_2|}{|\lambda_1|} e_k.$$

  We will later learn that this is linear convergence. It's quite slow.

  What does a shift do to this situation?

  $$e_{k+1} \approx \frac{|\lambda_2 - \sigma|}{|\lambda_1 - \sigma|} e_k.$$

  Picking $\sigma \approx \lambda_1$ does not help...

  Idea: Invert *and* shift to bring $|\lambda_1 - \sigma|$ into numerator.

Inverse Iteration / Rayleigh Quotient Iteration

- Describe inverse iteration.

$$\boldsymbol{x}_{k+1} := (A - \sigma)^{-1} \boldsymbol{x}_k$$

  - Implemented by storing/solving with LU factorization
  - Converges to eigenvector for eigenvalue closest to σ, with

$$e_{k+1} \approx \frac{|\lambda_{\text{closest}} - \sigma|}{|\lambda_{\text{second-closest}} - \sigma|} e_k.$$

- Describe Rayleigh Quotient Iteration.

  Compute $\sigma_k = \boldsymbol{x}_k^T A \boldsymbol{x}_k / \boldsymbol{x}_k^T \boldsymbol{x}_k$ to be the Rayleigh quotient for $\boldsymbol{x}_k$.

$$\boldsymbol{x}_{k+1} := (A - \sigma_k I)^{-1} \boldsymbol{x}_k$$

Demo: Power Iteration and its Variants

In-class activity: Eigenvalue Iterations

Computing Multiple Eigenvalues

- All Power Iteration Methods compute one eigenvalue at a time.
  What if I want *all* eigenvalues?

  Two ideas:

  1. Deflation: Suppose $A\boldsymbol{v} = \lambda \boldsymbol{v}$ ($\boldsymbol{v} \neq \boldsymbol{0}$). Let $V = \operatorname{span}\{\boldsymbol{v}\}$. Then

$$
\begin{aligned}
A\colon \quad V &\to V \\
V^{\perp} &\to V \oplus V^{\perp}
\end{aligned}
$$

     In matrix form

$$
A = \underbrace{\begin{pmatrix} | & & \\ \boldsymbol{v} & \text{Basis of } V^{\perp} & \\ | & & \end{pmatrix}}_{Q_1} \begin{pmatrix} \lambda & * & * & * \\ 0 & * & * & * \\ \vdots & * & * & * \\ 0 & * & * & * \end{pmatrix} Q_1^T.
$$

Now call B the shaded part of the resulting matrix

$$\text{eigenvalues of } A = \text{eigenvalues of } B \cup \{\lambda\}.$$

I.e. we've reduced the rest of the problem to finding the eigenvalues of B–which is smaller $\rightarrow$ We have shrunkt he problem size, or 'deflated' the problem.

2. Iterate with multiple vectors simultaneously.

Simultaneous Iteration

- What happens if we carry out power iteration on multiple vectors simultaneously?

  Simultaneous Iteration:

  1. Start with $X_0 \in \mathbb{R}^{n \times p}$ ($p \leqslant n$) with (arbitrary) iteration vectors in columns
  2. $X_{k+1} = A X_k$

  Problems:

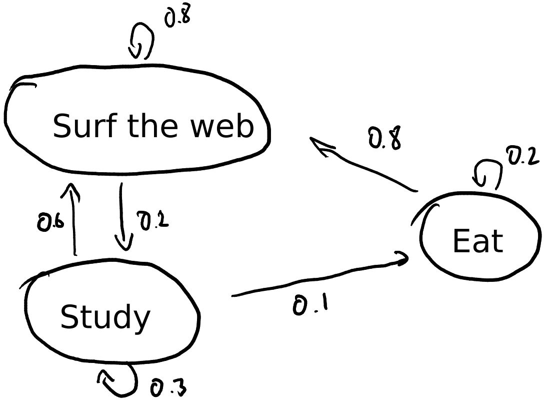
  - Needs rescaling
  - X increasingly ill-conditioned: all columns go towards $\boldsymbol{x}_1$

  Fix: orthogonalize! (using, e.g. Gram-Schmidt)

11 Eigenvalues: Applications

Markov chains

- Consider the following graph of states:



Suppose this is an accurate model of the behavior of the average student. :) How likely are we to find the average student in each of these states?

Important assumption: Only the most recent state matters to determine probability of next state. This is called the Markov property, and the model is called a Markov chain.

Write transition probabilities into matrix as before:
(Order: surf, study, eat–'from' state along columns)

$$A = \begin{pmatrix} .8 & .6 & .8 \\ .2 & .3 & 0 \\ 0 & .1 & .2 \end{pmatrix}$$

Recall: Columns add up to 1. Given probabilities of states $\boldsymbol{p} = (p_{\text{surf}}, p_{\text{study}}, p_{\text{eat}})$, $A\boldsymbol{p}$ gives us the probabilities after one unit of time has passed.

Idea: Look for a steady state, i.e. $A\boldsymbol{p} = \boldsymbol{p}$.

Phrase as an eigenvalue problem: $A\boldsymbol{p} = \lambda\boldsymbol{p}$.

Demo: Finding an equilibrium distribution using the power method

Understanding Time Behavior

- Many important systems in nature are modeled by describing the time rate of change of something.
  - E.g. every bird will have 0.2 baby birds on average per year.
  - But there are also foxes that eat birds. Every fox present decreases the bird population by 1 birds a year.

    Meanwhile, each fox has 0.3 fox babies a year. And for each bird present, the population of foxes grows by 0.9 foxes for every bird present.

Set this up as equations and see if eigenvalues can help us understand what's going on.

Equation just for birds:

$$\frac{\mathrm{d}}{\mathrm{d}t} b = 0.2b.$$

Equations for birds and foxes:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}b &= 0.2b - 1f\,,\\ \frac{\mathrm{d}}{\mathrm{d}t}f &= 0.9b + .3f\,.\end{aligned}$$

Shorter, letting the population $\boldsymbol{p} = (b, f)$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{p} = \begin{pmatrix} 0.2 & -1 \\ 0.9 & .3 \end{pmatrix}\boldsymbol{p}.$$

Bold (but pretty good) assumption:

$$\boldsymbol{p}(t) = e^{\lambda t}\boldsymbol{p}_0.$$

Then:

$$\lambda\boldsymbol{p}_0 = \begin{pmatrix} 0.2 & -1 \\ 0.9 & .3 \end{pmatrix}\boldsymbol{p}_0.$$

Hey look, an eigenvalue problem! :)

Demo: Understanding the birds and the foxes with eigenvalues
In-class activity: Eigenvalues 2

12 Approximate Undo: SVD and Least Squares

Singular Value Decomposition

- What is the Singular Value Decomposition ('SVD')?
  The SVD is a factorization of an $m \times n$ matrix into

$$A = U \Sigma V^T, \quad \text{where}$$

  - U is an $m \times m$ orthogonal matrix
    (Its columns are called 'left singular vectors'.)
  - Σ is an $m \times n$ diagonal matrix
    with the singular values on the diagonal

$$\Sigma = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & 0 \end{pmatrix} \qquad \text{Convention: } \sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_n \geqslant 0.$$

  - V^T is an $n \times n$ orthogonal matrix
    (V's columns are called 'right singular vectors'.)

Computing the SVD

- How can I compute an SVD of a matrix A?

1. Compute the eigenvalues and eigenvectors of $A^T A$.

$$A^T A \boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \quad \cdots \qquad A^T A \boldsymbol{v}_n = \lambda_n \boldsymbol{v}_n$$

2. Make a matrix V from the vectors $\boldsymbol{v}_i$:

$$V = \begin{pmatrix} | & & | \\ \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_n \\ | & & | \end{pmatrix}.$$

   ($A^T A$ symmetric: V orthogonal if columns have norm 1.)

3. Make a diagonal matrix Σ from the square roots of the eigenvalues:

$$\Sigma = \begin{pmatrix} \sqrt{\lambda_1} & & & \\ & \ddots & & \\ & & \sqrt{\lambda_n} & 0 \end{pmatrix}$$

4. Find U from

$$A = U\Sigma V^T \quad \Leftrightarrow \quad U\Sigma = AV.$$

(While being careful about non-squareness and zero singular values)
In the simplest case:

$$U = AV\Sigma^{-1}.$$

Observe U is orthogonal: (Use: $V^TA^TAV = \Sigma^2$)

$$U^TU = \Sigma^{-1}\underbrace{V^TA^TAV}_{\Sigma^2}\Sigma^{-1} = \Sigma^{-1}\Sigma^2\Sigma^{-1} = I.$$

(Similar for UU^T.)

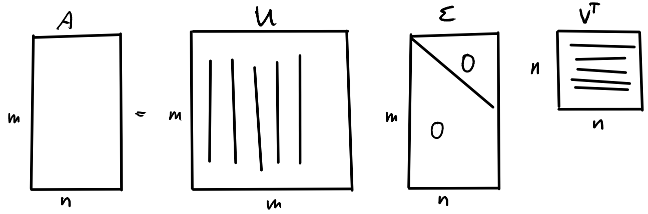
Demo: Computing the SVD

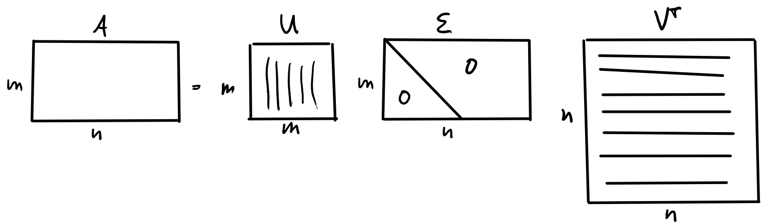
How Expensive is it to Compute the SVD?

Demo: Relative Cost of Matrix Factorizations

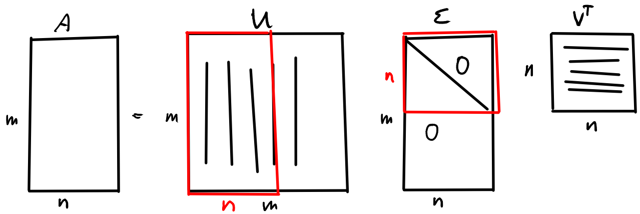
'Reduced' SVD

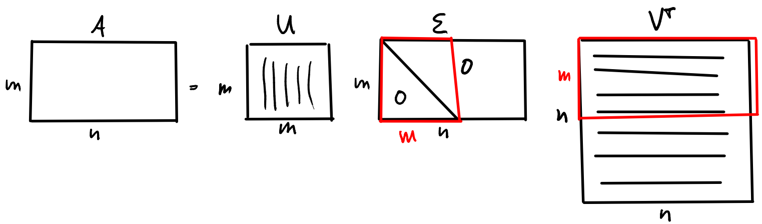
- Is there a 'reduced' factorization for non-square matrices?





Yes:





- "Full" version shown in black
- "Reduced" version shown in red

13 SVD: Applications

13.1 Solving Funny-Shaped Linear Systems

Solve Square Linear Systems

- Can the SVD $A = U\Sigma V^T$ be used to solve *square* linear systems? At what cost (once the SVD is known)?

  Yes, easy:

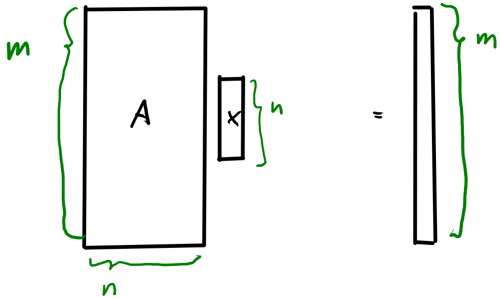
$$\begin{aligned} A\boldsymbol{x} &= \boldsymbol{b} \\ U\Sigma V^T\boldsymbol{x} &= \boldsymbol{b} \\ \Sigma \underbrace{V^T\boldsymbol{x}}_{\boldsymbol{y}:=} &= U^T\boldsymbol{b} \\ \text{(diagonal, easy to solve)} \quad \Sigma\boldsymbol{y} &= U^T\boldsymbol{b} \end{aligned}$$

  Know $\boldsymbol{y}$, find $\boldsymbol{x} = V\boldsymbol{y}$.

  Cost: $O(n^2)$–but overall much slower than using fw/bw subst. Even worse when including comparison of LU vs. SVD.

Tall and Skinny Systems

- Consider a 'tall and skinny' linear system, i.e. one that has more equations than unknowns:



In the figure: $m > n$. How could we solve that?

First realization: A square linear system often only has a single solution. So applying *more* conditions to the solution will mean we have no exact solution.

$$A\boldsymbol{x} = \boldsymbol{b} \quad \leftarrow \quad \text{Not going to happen.}$$

Instead: Find $\boldsymbol{x}$ so that $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is as small as possible.

$\boldsymbol{r} = A\boldsymbol{x} - \boldsymbol{b}$ is called the residual of the problem.

$$\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 = r_1^2 + \cdots + r_m^2 \quad \leftarrow \quad \text{squares}$$

This is called a (linear) least-squares problem. Since

Find $\boldsymbol{x}$ so that $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is as small as possible.

is too long to write every time, we introduce a shorter notation:

$$A\boldsymbol{x} \cong \boldsymbol{b}.$$

Solving Least-Squares

- How can I actually *solve* a least-squares problem $A\boldsymbol{x} \cong \boldsymbol{b}$?

  The job: Make $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is as small as possible.
  Equivalent: Make $\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2$ is as small as possible.
  Use: The SVD $A = U\Sigma V^T$.

  Find $\boldsymbol{x}$ to minimize:

$$
\begin{aligned}
& \|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 \\
= \; & \|U\Sigma V^T\boldsymbol{x} - \boldsymbol{b}\|_2^2 \\
= \; & \|U^T(U\Sigma V^T\boldsymbol{x} - \boldsymbol{b})\|_2^2 \qquad \text{(because } U \text{ is orthogonal)} \\
= \; & \left\| \Sigma \underbrace{V^T\boldsymbol{x}}_{\boldsymbol{y}} - U^T\boldsymbol{b} \right\|_2^2 \\
= \; & \|\Sigma\boldsymbol{y} - U^T\boldsymbol{b}\|_2^2
\end{aligned}
$$

What $\boldsymbol{y}$ minimizes

$$\|\Sigma \boldsymbol{y} - U^T \boldsymbol{b}\|_2^2 = \left\| \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_k & & \\ & & & 0 & \\ & & & & 0 \end{pmatrix} \boldsymbol{y} - \boldsymbol{z} \right\|_2^2 ?$$

Pick

$$y_i = \begin{cases} z_i / \sigma_i & \text{if } \sigma_i \neq 0, \\ 0 & \text{if } \sigma_i = 0. \end{cases}$$

Find $\boldsymbol{x} = V\boldsymbol{y}$, done.

Slight technicality: There only *is* a choice if some of the σ_i are zero. (Otherwise $\boldsymbol{y}$ is uniquely determined.) If there *is* a choice, this $\boldsymbol{y}$ is the one with the smallest 2-norm that *also* minimizes the 2-norm of the residual. And since $\|\boldsymbol{x}\|_2 = \|\boldsymbol{y}\|_2$ (because V is orthogonal), $\boldsymbol{x}$ *also* has the smallest 2-norm of all $\boldsymbol{x}'$ for which $\|A\boldsymbol{x}' - \boldsymbol{b}\|_2$ is minimal.

In-class activity: SVD and Least Squares

The Pseudoinverse: A Shortcut for Least Squares

- How could the solution process for $A\boldsymbol{x} \cong \boldsymbol{b}$ be with an SVD$A = U\Sigma V^T$ be 'packaged up'?

$$\begin{aligned} U\Sigma V^T \boldsymbol{x} &\approx \boldsymbol{b} \\ \Leftrightarrow \quad \boldsymbol{x} &\approx V\Sigma^{-1}U^T\boldsymbol{b} \end{aligned}$$

Problem: Σ may not be invertible.
Idea 1: Define a 'pseudo-inverse' Σ^+ of a diagonal matrix Σ as

$$\Sigma_i^+ = \begin{cases} \sigma_i^{-1} & \text{if } \sigma_i \neq 0, \\ 0 & \text{if } \sigma_i = 0. \end{cases}$$

Then $A\boldsymbol{x} \cong \boldsymbol{b}$ is solved by $V\Sigma^+U^T\boldsymbol{b}$.

Idea 2: Call $A^+ = V\Sigma^+U^T$ the pseudo-inverse of A.
Then $A\boldsymbol{x} \cong \boldsymbol{b}$ is solved by $A^+\boldsymbol{b}$.

The Normal Equations

- You may have learned the 'normal equations' $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$ to solve $A\boldsymbol{x} \cong \boldsymbol{b}$. Why not use those?

  $$\operatorname{cond}(A^T A) \approx \operatorname{cond}(A)^2$$

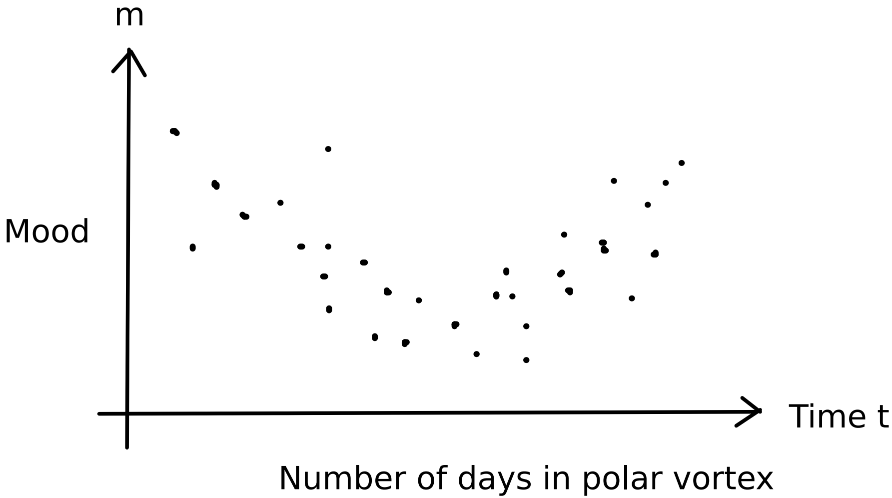
  I.e. if A is even somewhat poorly conditioned, then the conditioning of $A^T A$ will be a disaster.

  The normal equations are not well-behaved numerically.

13.2 Data Fitting

Fitting a Model to Data

- How can I fit a model to measurements? E.g.:



Maybe:

$$\hat{m}(t) = \alpha + \beta t + \gamma t^2$$

Have: 300 data pooints: $(t_1, m_1), \ldots, (t_{300}, m_{300})$
Want: 3 unknowns α, β, γ

Write down equations:

$$\begin{array}{rcl} \alpha + \beta t_1 + \gamma t_1^2 & \approx & m_1 \\ \alpha + \beta t_2 + \gamma t_2^2 & \approx & m_2 \\ \vdots & \vdots & \vdots \\ \alpha + \beta t_{300} + \gamma t_{300}^2 & \approx & m_{300} \end{array} \rightarrow \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \vdots \\ 1 & t_{300} & t_{300}^2 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \cong \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_{300} \end{pmatrix}.$$

So data fitting is just like interpolation, with a Vandermonde matrix:

$$V\boldsymbol{\alpha} = \boldsymbol{m}.$$

Only difference: More rows. Solvable using the SVD.

Demo: Data Fitting using Least Squares

13.3 Norms and Condition Numbers

Meaning of the Singular Values

- What do the singular values mean? (in particular the first/largest one)

  $A = U\Sigma V^T$

$$\begin{aligned}\|A\|_2 &= \max_{\|\boldsymbol{x}\|_2=1} \|A\boldsymbol{x}\|_2 = \max_{\|\boldsymbol{x}\|_2=1} \|U\Sigma V^T\boldsymbol{x}\|_2 \overset{U \text{ orth.}}{=} \max_{\|\boldsymbol{x}\|_2=1} \|\Sigma V^T\boldsymbol{x}\|_2 \\ &\overset{V \text{ orth.}}{=} \max_{\|V^T\boldsymbol{x}\|_2=1} \|\Sigma V^T\boldsymbol{x}\|_2 \overset{\text{Let } \boldsymbol{y}=V^T\boldsymbol{x}}{=} \max_{\|\boldsymbol{y}\|_2=1} \|\Sigma\boldsymbol{y}\|_2 \overset{\Sigma \text{ diag.}}{=} \sigma_1.\end{aligned}$$

  So the SVD (finally) provides a way to find the 2-norm.

  Entertainingly, it does so by reducing the problem to finding the 2-norm of a diagonal matrix.

$$\|A\|_2 = \sigma_1.$$

Condition Numbers

- How would you compute a 2-norm condition number?

$$\text{cond}_2(A) = \|A\|_2 \|A^{-1}\|_2 = \sigma_1 / \sigma_n.$$

13.4 Low-Rank Approximation

SVD as Sum of Outer Products

- What's another way of writing the SVD?

  Starting from (assuming $m > n$ for simplicity)

$$A = U\Sigma V^T = \begin{pmatrix} | & & | \\ \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_m \\ | & & | \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_m \\ & & 0 \end{pmatrix} \begin{pmatrix} - & \boldsymbol{v}_1 & - \\ & \vdots & \\ - & \boldsymbol{v}_n & - \end{pmatrix}$$

  we find that

$$\begin{aligned} A &= \begin{pmatrix} | & & | \\ \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_m \\ | & & | \end{pmatrix} \begin{pmatrix} - & \sigma_1 \boldsymbol{v}_1 & - \\ & \vdots & \\ - & \sigma_n \boldsymbol{v}_n & - \end{pmatrix} \\ &= \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T + \cdots + \sigma_n \boldsymbol{u}_n \boldsymbol{v}_n^T. \end{aligned}$$

  That means: The SVD writes the matrix A as a sum of outer products (of left/right singular vectors). What could that be good for?

Low-Rank Approximation (I)

- What is the *rank* of $\sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T$?

  1. (1 linearly independent column!)

- What is the *rank* of $\sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T$?

  2. (2 linearly independent–orthogonal–columns!)

Demo: Image Compression

Low-Rank Approximation

- What can we say about the low-rank approximation

$$A_k = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

to

$$A = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T + \cdots + \sigma_1 \boldsymbol{u}_n \boldsymbol{v}_n^T?$$

For simplicity, assume $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_n > 0$.
Observe that A_k has *rank* k. (And A has rank n.)

Then $\|A - B\|_2$ among all rank-k (or lower) matrices B is *minimized* by A_k.
(Eckart-Young Theorem)

Even better:

$$\min_{\text{rank } B \leqslant k} \|A - B\|_F = \|A - A_k\|_2 = \sigma_{k+1}.$$

A_k is called the best rank-k approximation to A.

(where k can be any number)

This best-approximation property is what makes the SVD extremely useful in applications and ultimately justifies its high cost.

It's also the rank-k best-approximation in the Frobenius norm:

$$\min_{\text{rank } B \leqslant k} \|A - B\|_F = \|A - A_k\|_F = \sqrt{\sigma_{k+1}^2 + \cdots \sigma_n^2}.$$

[Spring 17: Add PCA]

Part 3:
Approximation–When the Exact Answer is Out of Reach

14 Iteration and Convergence

- What is linear convergence? quadratic convergence?

  Let $\boldsymbol{e}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}$ be the error in the kth estimate $\hat{\boldsymbol{x}}_k$ of a desired solution $\boldsymbol{x}$.

  An iterative method converges with rate r if

$$\lim_{k\to\infty} \frac{\|\boldsymbol{e}_{k+1}\|}{\|\boldsymbol{e}_k\|^r} = C \begin{cases} >0, \\ <\infty. \end{cases}$$

  $r = 1$ is called linear convergence.
  $r > 1$ is called superlinear convergence.
  $r = 2$ is called quadratic convergence.

  Examples:

  - Power iteration is linearly convergent.
  - Rayleigh quotient iteration is quadratically convergent.

About Convergence Rates

Demo: Rates of Convergence

- Characterize linear, quadratic convergence in terms of the 'number of accurate digits'.
  - Linear convergence gains a constant number of digits each step:

    $$\|\boldsymbol{e}_{k+1}\| \leqslant C \|\boldsymbol{e}_k\|$$

    (and $C < 1$ matters!)
  - Quadratic convergence doubles the number of digits each step:

    $$\|\boldsymbol{e}_{k+1}\| \leqslant C \|\boldsymbol{e}_k\|^2$$

    (Only starts making sense once $\|\boldsymbol{e}_k\|$ is small. C doesn't matter much.)

15 Solving One Equation

Solving Nonlinear Equations

- What is the goal here?

  Solve $f(x) = 0$ for $f: \mathbb{R} \to \mathbb{R}$.

  If looking for solution to $f(x) = y$, simply consider $f(x) = \tilde{f}(x) - y$.

  Intuition: Each of the n equations describes a surface. Looking for intersections.

Bisection Method

Demo: Bisection Method

- What's the rate of convergence? What's the constant?
  Linear with constant $1/2$.

Newton's Method

- Derive Newton's method.

  Idea: Approximate f at current iterate using Taylor.

  $$f(x_k+h)\approx f(x_k)+f'(x_k)h$$

  Now find root of this linear approximation in terms of h:

  $$f(x_k)+f'(x_k)h=0 \quad \Leftrightarrow \quad h=-\frac{f(x_k)}{f'(x_k)}.$$

  So

  $$\begin{aligned} x_0 &= \langle \text{starting guess} \rangle \\ x_{k+1} &= x_k-\frac{f(x_k)}{f'(x_k)}=g(x_k) \end{aligned}$$

Demo: Newton's method
Demo: Convergence of Newton's Method

- What are some **drawbacks** of Newton?

  - Convergence argument only good *locally*
    Will see: convergence only local (near root)

  - Have to have derivative!

Secant Method

- What would Newton without the use of the derivative look like?

  Approximate

  $$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

  So

  $$\begin{aligned} x_0 &= \langle \text{starting guess} \rangle \\ x_{k+1} &= x_k - \frac{f(x_k)}{\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}}. \end{aligned}$$

  Rate of convergence (not shown) is $\left(1 + \sqrt{5}\right)/2 \approx 1.618$.

  Drawbacks of Secant:

  - Convergence argument only good *locally*
    Will see: convergence only local (near root)

- Slower convergence
- Need two starting guesses

Demo: Secant Method

In-class activity: Nonlinear equations in 1D

16 Solving Many Equations

Solving Nonlinear Equations

- What is the goal here?

  Solve $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$ for $f : \mathbb{R}^n \to \mathbb{R}^n$.

  If looking for solution to $\tilde{\boldsymbol{f}}(\boldsymbol{x}) = \boldsymbol{y}$, simply consider $\boldsymbol{f}(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}) - \boldsymbol{y}$.

  Intuition: Each of the n equations describes a surface. Looking for intersections.
  Demo: Intersection of quadratics

Newton's method

- What does Newton's method look like in n dimensions?

  Approximate by linear function:

  $$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{s}) = \boldsymbol{f}(\boldsymbol{x}) + J_{\boldsymbol{f}}(\boldsymbol{x})\boldsymbol{s}$$

  where J_f is the Jacobian matrix of f:

  $$J_f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{\boldsymbol{n}}} \\ \vdots & \ddots & \\ \frac{\partial f_{\boldsymbol{n}}}{\partial x_1} & \cdots & \frac{\partial f_{\boldsymbol{n}}}{\partial x_{\boldsymbol{n}}} \end{pmatrix}(\boldsymbol{x}).$$

  Set to $\mathbf{0}$:

  $$J_{\boldsymbol{f}}(\boldsymbol{x})\boldsymbol{s} = -\boldsymbol{f}(\boldsymbol{x}) \quad \Rightarrow \quad \boldsymbol{s} = -(J_{\boldsymbol{f}}(\boldsymbol{x}))^{-1}\boldsymbol{f}(\boldsymbol{x})$$

  That's a linear system! (Surprised?)

So

$$\begin{aligned} \boldsymbol{x}_0 &= \langle \text{starting guess} \rangle \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - (J_{\boldsymbol{f}}(\boldsymbol{x}_k))^{-1} \boldsymbol{f}(\boldsymbol{x}_k) \end{aligned}$$

Downsides:

- Still only locally convergent
- Computing and inverting $J_{\boldsymbol{f}}$ is expensive.

Newton: Example

- Set up Newton's method to find a root of

$$f(x,y) = \begin{pmatrix} x + 2y - 2 \\ x^2 + 4y^2 - 4 \end{pmatrix}.$$

Mostly just need the Jacobian:

$$J_f(x,y) = \begin{pmatrix} 1 & 2 \\ 2x & 8y \end{pmatrix}.$$

Demo: Newton's method in n dimensions

Secant in n dimensions?

- What would the secant method look like in n dimensions?

  Need an 'approximate Jacobian' satisfying

  $$\tilde{J}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \boldsymbol{f}(\boldsymbol{x}_{k+1}) - \boldsymbol{f}(\boldsymbol{x}_k).$$

  Suppose we have *already taken* a step to $\boldsymbol{x}_{k+1}$. Could we 'reverse engineer' $\tilde{J}$ from that equation?

  - No: n^2 unknowns in $\tilde{J}$, but only n equations
  - Can only hope to 'update' $\tilde{J}$ with information from current guess.

  One choice: Broyden's method (minimizes change to $\tilde{J}$)

17 Finding the Best: Optimization in 1D

Optimization

- State the problem.

  Have: Objective function $f : \mathbb{R}^n \to \mathbb{R}$
  Want: Minimizer $\boldsymbol{x}^* \in \mathbb{R}^n$ so that

  $$f(\boldsymbol{x}^*) = \min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to} \quad \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{h}(\boldsymbol{x}) \leqslant \boldsymbol{0}.$$

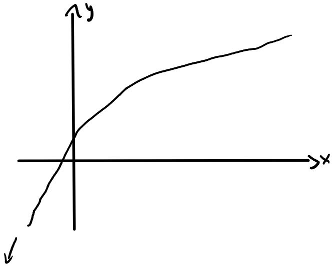
  - $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ and $\boldsymbol{h}(\boldsymbol{x}) \leqslant \boldsymbol{0}$ are called constraints.
    They define the set of feasible points $\boldsymbol{x} \in S \subseteq \mathbb{R}^n$.
  - If $\boldsymbol{g}$ or $\boldsymbol{h}$ are present, this is constrained optimization.
    Otherwise unconstrained optimization.
  - If $\boldsymbol{f}$, $\boldsymbol{g}$, $\boldsymbol{h}$ are *linear*, this is called linear programming.
    Otherwise nonlinear programming.
  - **Q:** What if we are looking for a maximizer?
    A: Minimize $-f$ instead.

- Examples:
    - What is the fastest/cheapest/shortest... way to do...?
      Q: What about multiple objectives?
      A: Make up your mind–decide on one (or build a combined objective). Then we'll talk.
    - Solve a (nonlinear!) system of equations 'as well as you can' (if no exact solution exists)–similar to what least squares does for linear systems:

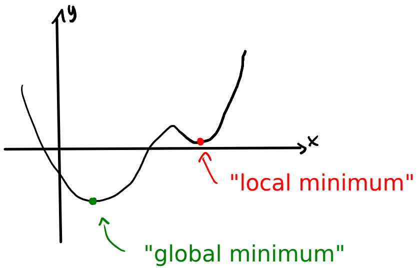
$$\min \|F(\boldsymbol{x})\|$$

Optimization: What could go wrong?

- What are some potential problems in optimization?
  - No minimum exists: Function just 'keeps going'.



  - Find a local minimum when we meant to find a global minimum.



Optimization: What is a solution?

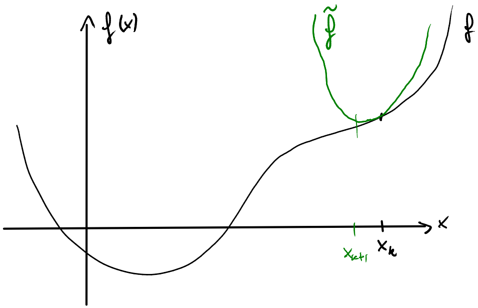
- How can we tell that we have a (at least local) minimum? (Remember calculus!)
  - Necessary condition: $f'(x) = 0$
  - Sufficient condition: $f'(x) = 0$ *and* $f''(x) > 0$.

Newton's Method

- Let's steal the idea from Newton's method for equation solving:
  Build a simple version of f and minimize that.

  Use Taylor approximation–with what degree?

  Note: Line (i.e. degree 1 Taylor) wouldn't suffice–lines have no minimum. Must use at least parabola. (degree 2)



$$f(x+h) \approx f(x) + f'(x)h + f''(x)\frac{h^2}{2} =: \tilde{f}(h)$$

Solve $0 = \tilde{f}'(h) = f'(x) + f''(x)h$:

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

1. $x_0 = \langle \text{some starting guess} \rangle$
2. $x_{k+1} = x_k - \frac{f'(x_{\boldsymbol{k}})}{f''(x_{\boldsymbol{k}})}$

Q: Notice something? Identical to Newton for solving $f'(x) = 0$.
Because of that: locally quadratically convergent.

Demo: Newton's method in 1D

In-class activity: Optimization Methods

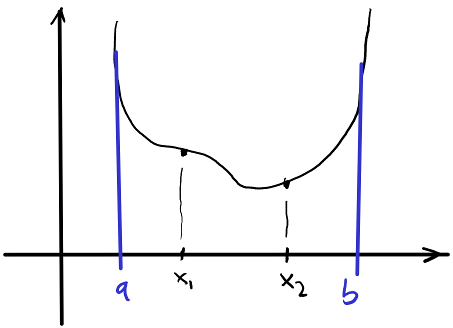
Golden Section Search [Not on the exam]

- Would like a method like bisection, but for optimization.
  In general: No invariant that can be preserved.
  Need *extra assumption.*

  f is called unimodal if for all $x_1 < x_2$

  - $x_2 < x^* \Rightarrow f(x_1) > f(x_2)$
  - $x^* < x_1 \Rightarrow f(x_1) < f(x_2)$

  Suppose we have an interval with f unimodal:



Would like to maintain unimodality.

1. Pick x_1, x_2
2. If $f(x_1) > f(x_2)$, reduce to (x_1, b)
3. If $f(x_1) \leqslant f(x_2)$, reduce to (a, x_2)

Remaining question: Where to put x_1, x_2?

- Want symmetry:
  $x_1 = a + (1 - \tau)(b - a)$
  $x_2 = a + \tau(b - a)$
- Want to reuse function evaluations: $\tau^2 = 1 - \tau$
  Find: $\tau = \left(\sqrt{5} - 1\right)/2$. Also known as the 'golden section'.
- Hence golden section search.

Linearly convergent. Can we do better?

Demo: Golden Section Search Proportions

18 Optimization in n Dimensions

Optimization in n dimensions: What is a solution?

- How can we tell that we have a (at least local) minimum? (Remember calculus!)
  - Necessary condition: $\nabla f(\boldsymbol{x}) = 0$
    ∇f is a vector, the gradient:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

  - Sufficient condition: $\nabla f(\boldsymbol{x}) = 0$ *and* $H_f(\boldsymbol{x})$ positive definite.

$$H_f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

is called the Hessian matrix.

Steepest Descent [Not on the exam]

- Given a scalar function $f: \mathbb{R}^n \to \mathbb{R}$ at a point $\boldsymbol{x}$, which way is down?

  Direction of steepest descent: $-\nabla f$

  Q: How far along the gradient should we go?

  Unclear–do a line search. For example using Golden Section Search.

  1. $\boldsymbol{x}_0 = \langle$some starting guess$\rangle$
  2. $\boldsymbol{s}_k = -\nabla f(\boldsymbol{x}_k)$
  3. Use line search to choose α_k to minimize $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k)$
  4. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$
  5. Go to 2.

  Observation: (from demo)

  - Linear convergence

Demo: Steepest Descent

Newton's method (nD)

- What does Newton's method look like in n dimensions?

  Build a Taylor approximation:

  $$f(\boldsymbol{x}+\boldsymbol{s}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T H_f(\boldsymbol{x})\boldsymbol{s} =: \hat{f}(\boldsymbol{s})$$

  Then solve $\nabla \hat{f}(\boldsymbol{s}) = \mathbf{0}$ for $\boldsymbol{s}$ to find

  $$H_f(\boldsymbol{x})\boldsymbol{s} = -\nabla f(\boldsymbol{x}).$$

  1. $x_0 = \langle \text{some starting guess} \rangle$
  2. Solve $H_f(\boldsymbol{x}_k)\boldsymbol{s}_k = -\nabla f(\boldsymbol{x}_k)$ for $\boldsymbol{s}_k$
  3. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$

  Drawbacks: (from demo)

  - Need second (!) derivatives

(addressed by Conjugate Gradients, later in the class)

- local convergence
- Works poorly when H_f is nearly indefinite

Demo: Newton's method in n dimensions

Demo: Nelder-Mead Method

Nonlinear Least Squares/Gauss-Newton

- What if the f to be minimized is actually a 2-norm?

$$f(\boldsymbol{x}) = \|\boldsymbol{r}(\boldsymbol{x})\|_2, \qquad \boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x})$$

Define 'helper function'

$$\varphi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{r}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}) = \frac{1}{2}f^2(\boldsymbol{x})$$

and minimize that instead.

$$\frac{\partial}{\partial x_i}\varphi = \frac{1}{2}\sum_{j=1}^{n}\frac{\partial}{\partial x_i}[r_j(\boldsymbol{x})^2] = \sum_j \left(\frac{\partial}{\partial x_i}r_j\right)r_j,$$

or, in matrix form:

$$\nabla\varphi = J_{\boldsymbol{r}}(\boldsymbol{x})^T\boldsymbol{r}(\boldsymbol{x}).$$

For brevity: $J := J_{\boldsymbol{r}}(\boldsymbol{x})$. Can show similarly:

$$H_\varphi(\boldsymbol{x}) = J^T J + \sum_i r_i H_{r_i}(\boldsymbol{x}).$$

Newton step $\boldsymbol{s}$ can be found by solving

$$H_\varphi(\boldsymbol{x})\boldsymbol{s} = -\nabla\varphi$$

Observation: $\sum_i r_i H_{r_i}(\boldsymbol{x})$ is inconvenient to compute *and* unlikely to be large (since it's multiplied by components of the residual, which is supposed to be small) $\rightarrow$ forget about it.

Gauss-Newton method: Find step $\boldsymbol{s}$ by

$$J^T J\boldsymbol{s} = -\nabla\varphi = -J^T\boldsymbol{r}(\boldsymbol{x})$$

Does that remind you of the *normal equations*?

$$J\boldsymbol{s} \cong -\boldsymbol{r}(\boldsymbol{x})$$

Solve that using our existing methods for least-squares problems.

Observations: (from demo)

- Newton on its own is only locally convergent
- Gauss-Newton is clearly similar
- It's worse because the step is only approximate
  $\rightarrow$ Much depends on the starting guess.

If Gauss-Newton on its own is poorly, conditioned, can try **Levenberg-Marquardt**:

$$(J_{\boldsymbol{r}}(\boldsymbol{x}_k)^T J_{\boldsymbol{r}}(\boldsymbol{x}_k) + \mu_k I)\boldsymbol{s}_k = -J_{\boldsymbol{r}}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k)$$

for a 'carefully chosen' μ_k. This makes the system matrix 'more invertible' but also less accurate/faithful to the problem. Can also be translated into a least squares problem (see book).

What Levenberg-Marquardt does is generically called 'Regularization': Make H more positive definite.

Demo: Gauss-Newton