24310 - Numerical Linear Algebra - Week 1

Tuesday - 03/21/23

- Course Info:
  - Instructor: Alex Strang, alexstrang@uchicago.edu
    - Office: Jones 309
  - TA: Hwanwoo (Josh) Kim, hwkim@uchicago.edu
  - Office Hours: Wednesdays, 9:00 - 10:30 am, Jones 304
  - Lab Sessions: Fridays, 3:00 - 5:00 pm, Jones 226
  - Materials: Numerical Linear Algebra by Trefethen and Bau
  - Assignments and projects in Jupyter note book (install anaconda locally)
  - All info, announcements, calendar, etc. on canvas

- Logistics:
  - Read part I and lectures 12-15 of part III
  - Complete course poll
  - Install anaconda and make sure you can launch Jupyter notebook

- Goals:
  - Intro to numerical analysis/scientific computing
    - Problem type, accuracy/stability, cost/efficiency
      - floating point arithmetic
      - Stability
      - Conditioning
      - Big O notation

      } accuracy \$ stability

- What is numerical analysis?
  - the study of methods/algorithms for continuous problems

    methods/algorithms: computation, numerics

    continuous: focus on scientific problems, statistical/data science problems that involve continuous variables

  - Computation: is discrete and finite

- the tension between continuous problems $ discrete methods will raise the Key issues of the feild
  - accuracy
  - scope of what can be approximation
  - cost/efficiency
  - convergence
  - discretization
  - rounding
  - error propagation
  - stability & robustness

- Key moral:
  - There are often many mathematically equivalent ways to solve a problem, that are numerically completely different.
  - How? how you solve a problem matters.

- Design Principles for Numerical Analysis: "axes of analysis"

  1. What are you solving?
     a. what is the general problem class?
     b. are there specific features of the problem I can exploit?
        (i) symmetry
        (ii) convexity
        (iii) sparsity
        (iv) prior Knowledge or constraints on your soln's

  2. How accurately? real problems have errors:
     (i) errors in model, problem
     (ii) error in inputs that specify a problem instance } not really numerical analysis
     (iii) approximation errors from choice of method (errors from discretization)
     (iv) computational/rounding errors } numerical analysis
     - Achieve a desired accuracy. (depends on controlling all 4 sources)
     - Numerically, aim for stability ⇒ small relative errors in output, given small errors in input

  3. How quickly? what is your computational budget?
     (i) memory/storage
     (ii) # of operations needed
     (iii) clock time/wall time
     (iv) # of computational nodes, cost of the cluster, GPU, externalities

- **Ex:** given $A \in \mathbb{C}^{m \times n}$, $b \in \mathbb{C}^n$ find $x \in \mathbb{C}^m$ s.t. $Ax = b$.

  then you'd call $x = A \backslash b$.

  - **Q:** what does $A \backslash b$ actually <u>do</u> in a real comp. lang.
    - does it calculate $A^{-1} \Rightarrow A^{-1}b = x$? ** basically never
    - approx. A^{-1} and do the same thing?
    - reduce A or factor it, then solve via the factors?
      - $LU \Leftrightarrow$ Gaussian elimination
      - $QR \Leftrightarrow$ Gram-Schmidt

    direct methods
    - iterative methods based on optimization

- <u>Why linear algebra?</u>

  a. most problems are multivariate

  b. most* multivariate problems are hard

  c. * unless they are linear (~ linear)

  (b/c linearity allows decomposition into parts sequential reduction process, convert an n-D problem into a sequence of n, 1-D problems)

  d. and they are useful for approximation, appear as intermediate computational steps of other algorithms, and are themselves ubiquituous

  <u>Consequence:</u>

  if the basic operations done in a comp. system are $+, -, \times, \div$

  then the basic computation steps used in algorithms are linear algebraic.

- <u>The Basics:</u>
  - Number System $\Rightarrow$ Floating Point #'s.

  - problem: computers can't represent ∞-ly many #'s so they are restricted to a finite subset
    - subset we use are the floating point #'s

- Floating • #'s are the comp. analog to scientific notation

• **Def:** a set of floating point #'s F is defined by:

1. a base, β, integer valued, >1 (in practice $\beta=2$)
2. a precision, t, integer valued, >1 (in practice $t=24$, or $t=53$)

(3. a scale, E, integer valued, >1 ← fix the smallest & largest #'s in F, in practice the smallest & largest #'s in F are 10^{-308}, 10^{308})

then F is all #'s x s.t.

$$x = \pm\left(\frac{m}{\beta^t}\right)\beta^e \quad \text{for} \quad m \in [\beta^{t-1}, \beta^t - 1], \quad e \in [-E, E]$$

(idea: is to break the real line into powers of β, then separate those intervals evenly using segments spaced according to the precision. do this between two values controlled by $-E, E$.)

• **Ex:** $\beta=2$, $t=3$, $E=2$

Thursday - 03/23 - Computational Basics

- **Logistics:**
  - Course intro poll
  - Read Part I (Lin. Alg. review) & lectures 12-15
  - Install anaconda & open Jupyter notebook

Goals:

- Measuring Computational Cost & "big $\mathcal{O}$" notation
- Accuracy, Stability & Conditioning
  - Floating Point #'s and machine ε
  - Stability
  - Conditioning
- Norms
  - Vector norms
  - Matrix norms

Measuring Computational Cost

- numerics: # of flops - floating point operations $(+, -, \times, \div)$ required by an algorithm
  memory usage or storage, # of floating point values stored

  - often study the # of flops required in an asymptotic sense
  - how does the # of operations required scale w/ problem size?

  - "big $\mathcal{O}$" notation...
    - Ex: solve $Ax=b$ for $A \in \mathbb{C}^{n \times n}$, use a direct method, $\mathcal{O}(n^3)$
      means: # operations required (n) bounded from above by Cn^3, $C > 0$ for sufficiently large n

  - Def: a function $f(t)$, $t \in \mathbb{R}$ is $\mathcal{O}(g(t))$ if $\exists$ $C > 0$, $C \in \mathbb{R}$ s.t.

$$|f(t)| \leq C\, g(t), \quad g(t) > 0, \quad g(t) \in \mathbb{R}$$

(more generally, often meant in a limiting sense, either $\forall\, t \geq T$, or $\forall\, t \leq T$)

- this notion, measure cost of an algorithm via a bound on its scaling in problem size ⇒ complexity

  - problems arising from lin. alg. (those that admit direct methods) run in "polynomial time"

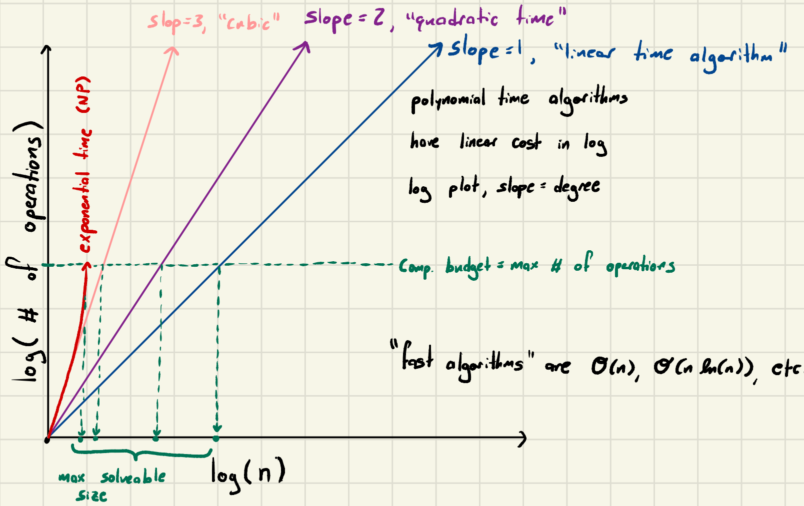
  - P is the class of polynomial time problems if all problems in P admit an algorithm that produces soln's using $\mathcal{O}(p(n))$ steps where $p(n)$ is a polynomial of finite degree, n = problem size

    - iterative methods are often derived from optimization usually thinking in terms of convergence analysis fix accuracy → complexity

  - NP is the class of nondeterministic polynomial time problems
    - there is no known polynomial time method for producing soln's
    - but any soln can be verified in polynomial time

  Ex: combinatorial optimization problems whose state space grows exponentially in problem size

  - usually think P = tractable at large scale (may be expensive)
    NP = intractable at large scale



polynomial time algorithms have linear cost in log log plot, slope = degree

"fast algorithms" are $\mathcal{O}(n)$, $\mathcal{O}(n\ln(n))$, etc.

Accuracy, Stability & Conditioning

- Floating · #'s: the comp. analog to scientific notation
  - problem: computers can't represent ∞-ly many #'s so they are restricted to a finite subset
    - subset we use are the floating point #'s

- Def: a set of floating point #'s F is defined by:
  1. a base, β, integer valued, >1 (in practice $\beta=2$)
  2. a precision, t, integer valued, >1 (in practice $t=24$, or $t=53$)

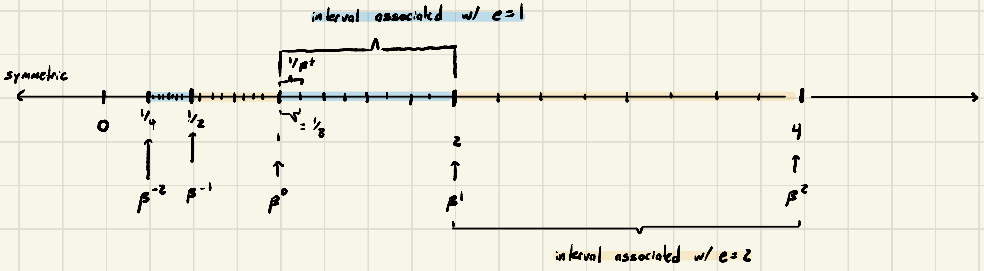
  (3. a scale, E, integer valued, >1 ← fix the smallest & largest #'s in F, in practice the smallest & largest #'s in F are 10^{-308}, 10^{308})

  then F is all #'s x s.t.

$$x = \pm\left(\frac{m}{\beta^t}\right)\beta^e \quad \text{for} \quad m \in [\beta^{t-1}, \beta^t - 1], \quad e \in [-E, E]$$

(idea: is to break the real line into powers of β, then separate those intervals evenly using segments spaced according to the precision. do this between two values controlled by $-E$, E.)

- Ex: $\beta=2$, $t=3$, $E=2$

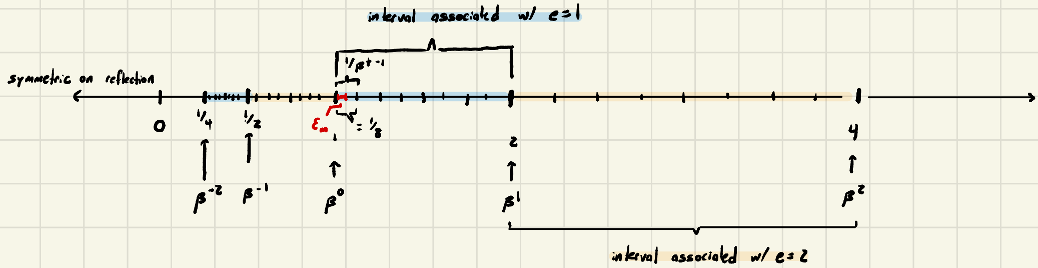


- check: if $m=\beta^{t-1}$ then $\frac{m}{\beta^t} = \beta^{-1}$ so $\left(\frac{m}{\beta^t}\right)\beta^e = \beta^{e-1}$ ← lower endpoint of the e interval

  if $m=\beta^t-1$ then $\frac{m}{\beta^t} = 1 - 1/\beta^t$ so $\left(\frac{m}{\beta^t}\right)\beta^e = \beta^e - \left(\frac{1}{\beta^t}\right)\beta^e$ ← last # before β^e

spacing in e^{th} interval: $\left(\frac{m+1}{\beta^t}\right)\beta^e - \left(\frac{m}{\beta^t}\right)\beta^e = \left(\frac{1}{\beta^t}\right)\beta^e$

so, in e^{th} interval, divide into evenly spaced segments of size $\left(\frac{1}{\beta^t}\right)\beta^e$



- What is the precision of calculations using F?

  - <u>Def:</u> <u>machine epsilon</u>, $\varepsilon_m = \frac{1}{2}\beta^{t-1}$, ($\frac{1}{2}$ the gap between 1, and the next largest # in F)

    - set a lower limit on tolerance of any calculation (in a relative sense)

      - IEEE double precision then $\varepsilon_m = 2^{-53} \approx 1.1 \times 10^{-16}$

  - <u>Lemma:</u> if $|x| \in [\min(F), \max(F)]$ then $\exists\ x' \in F$ s.t.

$$\frac{|x - x'|}{|x|} \leq \varepsilon_m \iff |x - x'| \leq \varepsilon_m |x|$$

and, if $fl : \mathbb{R} \to F$ rounding to the nearest # in F, then $\exists\ \varepsilon > 0$ s.t. $|\varepsilon| \leq \varepsilon_m$ and $fl(x) = x(1+\varepsilon)$

  - <u>Axiom of floating - arithmetic:</u> assume that our computers implement a base set of operations $(+, -, \times, \div)$ through floating point equivalents $(\oplus, \ominus, \dots)$ $\{ * \xrightarrow[fl]{} \circledast \}$ such that $\exists\ \varepsilon > 0$ w/ $|\varepsilon| \leq \varepsilon_m$ and

$$x \circledast y = (x * y)(1+\varepsilon)$$

    - can guarantee accuracy of the basic operations up to ε_m

- <u>Accuracy & Stability of problems</u>: problem instance, inputs x, a mapping, function f which returns desired outputs, i.e. soln's
  - let $\tilde{f}$ denote the approximate soln's produced by a computational routine

- given $f: X \to Y$, (X, Y are vector spaces) then:
  - space of inputs: X
  - outputs: Y

- <u>Def</u>: the relative accuracy of $\tilde{f}$ at x: $\dfrac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|}$

- from the numerical perspective, we design $\tilde{f}$, but don't have control over errors in the inputs → instead design for stability
  i.e. relatively small errors in the output provided small errors in the input

- <u>Stability</u>: given f, $\tilde{f}$ is <u>stable</u> if $\forall\ x \in X$, $\tilde{x}$ s.t. $\|\tilde{x} - x\| / \|x\| = \mathcal{O}(\varepsilon_m)$
  then:
  $$\|\tilde{f}(x) - f(\tilde{x})\| / \|f(\tilde{x})\| = \mathcal{O}(\varepsilon_m)$$

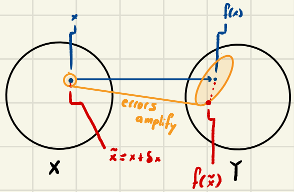
  "nearly the right answer to nearly the right problem"

- <u>Backward Stability</u>: given f, $\tilde{f}$ is <u>backward stable</u>, if $\forall\ x \in X$, $\exists\ \tilde{x}$ s.t.
  $$\tilde{f}(x) = f(\tilde{x}) \quad \text{where} \quad \|\tilde{x} - x\| / \|x\| = \mathcal{O}(\varepsilon_m)$$

  "exactly the right answer to nearly the right problem"

- <u>Conditioning</u>: whether / the degree to which stability is achievable depends on the problem we are trying to solve... sensitivity of outputs $f(x)$ to inputs x
  - the scaling factor "hidden" in big $\mathcal{O}(\varepsilon)$ statements
  - the # digits of accuracy lost by applying f, (given $\varepsilon_m \approx 10^{-16}$, only have 16 digits to lose)

- If f: inputs $\to$ outputs (instance $\to$ soln), sensitivity of $f(x)$ to perturbations in x will bound the stability of best possible $\tilde{f}$



- if f is highly sensitive to inputs $\Rightarrow \tilde{f}$ cannot be stable (lose accuracy) } "ill conditioned"
- if f is not $\Rightarrow \tilde{f}$ could retain most of digits of accuracy } "well-conditioned"

- Bound the sensitivity of f from above w/ regularity or smoothness conditions on f:
  - Ex: f is Lipschitz continuous X w/ constant K.

    then $\forall\ x, y \in X$

$$\|f(x) - f(y)\| \leq K\|x-y\| \Rightarrow \frac{\|f(x) - f(\tilde{x})\|}{\|x-\tilde{x}\|} \leq K$$

so, if $\|x - \tilde{x}\| = \mathcal{O}(\varepsilon_m) \Rightarrow \|f(x) - f(\tilde{x})\| = \mathcal{O}(K\varepsilon_m)$

- Bounds on sensitivity of f, from below, condition #, κ
  - well-cond. problems have κ small
  - ill-cond. problems have κ large

- Def: given a problem instance (f, x) an absolute condition #, $\hat{\kappa}(f,x)$ and a relative condition number $\kappa(f,x)$

$$\hat{\kappa}(f,x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|\delta f\|}{\|\delta x\|}$$

$$\kappa(f,x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \left(\frac{\|\delta f\|}{\|f\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right)$$

- $\kappa(f,x)$ worst case loss in relative accuracy at (f,x)

• **Ex:** let $f(x) = x_1 - x_2$ (problem: compute the difference in 2 inputs)

then (see Trefethen), $\kappa(f, x) = 2 \dfrac{\max\{|x_1|, |x_2|\}}{|x_1 - x_2|}$

$\therefore$ subtraction of two large, similar #'s is unstable

require high relative precision in input to retain precision in output

(important since rounding errors in fl. $\cdot$ arithmetic are relative

subtraction is unstable when 2 #'s nearly cancel)

- An example of different mathematically equivalent statements that are numerically distinct.

$$3 - 2 = 1, \quad 5 - 4 = 1, \quad 11 - 10 = 1 \rightarrow (3 + x) - (2 + x) = 1$$

but $\kappa(f, x) = 2 \cdot (3 + x) = O(x)$. $\quad (3 + x) - (2 + x) = 1 \ \forall x$

but get's less stable as x grows.

Week 2 - Linear Transformations

Tuesday - 03/28/23

- Logistics:
  - HW 1 posted, due next Tuesday (by midnight, on canvas)
  - lab sessions and office hours start this week
  - Read Part I, start Part II (lectures 6-8)

- Goals:
  - Norms
    - Vector norms
    - Operator norms
    - Conditioning revisited
  - Linear Transformations
    - Vector spaces and linear operations
    - Finite Dimensional Linear Transformations
    - Spectral radius & condition #'s

- Review: Conditioning

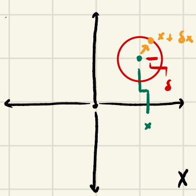
  - a problem is really a mapping f from inputs $x \to$ outputs $f(x)$
    computationally approximate $f(x)$ w/ $\tilde{f}(\tilde{x})$

    - Def: given a problem instance (f, x) and a norm, $\|\cdot\|$
      where $f: X \to Y$, X, Y are vector spaces
      an absolute condition number $\hat{\kappa}(f, x)$ and relative condition # $\kappa(f, x)$

worst case error amplification

$$\hat{\kappa}(f, x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \le \delta} \left\{ \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|} \right\}$$

$$\kappa(f, x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \le \delta} \left\{ \frac{\|f(x+\delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right\}$$



- Ex: let $f(x) = x_1 - x_2$ for $x \in \mathbb{R}^2$, $\|x\| = \max_j \{|x_j|\}$

  what is $\hat{\kappa}(f, x)$, $\kappa(f, x)$?

$$\hat{\kappa}(f,x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|} = \|J(x)\|_{?}$$

why? in what sense?

where $J(x) = \begin{bmatrix} \partial_{x_1} f_1(x) & \cdots & \partial_{x_n} f_1(x) \\ \vdots & & \vdots \\ \partial_{x_1} f_m(x) & \cdots & \partial_{x_n} f_m(x) \end{bmatrix}$, $f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}$

$f(x): \mathbb{R}^2 \to \mathbb{R}$, $f(x) = x_1 - x_2$, $J(x) = [1, -1]$

then using $\|x\| = \max_j \{|x_j|\}$, then $\|[1, -1]\|_{?} = |1| + |-1|$
$= 1 + 1 = 2$

$\hat{\kappa}(f,x) = 2$

$$\kappa(f,x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \left\{ \frac{\|f(x+\delta x) - f(x)\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right\} = \lim_{\delta \to 0} \sup_{\|\delta x\| \leq \delta} \left\{ \frac{\|f(x+\delta x) - f(x)\|}{\|\delta x\|} \right\} \frac{\|x\|}{\|f(x)\|}$$

$$= \hat{\kappa}(f,x) \frac{\|x\|}{\|f(x)\|} = 2 \frac{\max\{|x_1|, |x_2|\}}{|x_1 - x_2|}$$

- computing a difference is ill-conditioned if $\max\{|x_1|, |x_2|\} \gg |x_1 - x_2|$

  - Ex: compute $(x+1) - x = 1 \Rightarrow 1 - 0 = 1$
    $2 - 1 = 1$
    $11 - 10 = 1$
    $\vdots$
    mathematically equivalent but numerically distinct

$$\kappa(f,x) = 2 \max\{|x|, 1\} = \mathcal{O}(|x|)$$

- Norms: measuring size, how big is $x \in X$?

  - Def: given a vector space X, a norm $\|\cdot\|: X \to \mathbb{R}$, $\|x\|$ assigns a real # to any $x \in X$ such that
    1. $\|x\| \geq 0$ and $\|x\| = 0$ iff $x = 0$
    2. $\|\alpha x\| = |\alpha| \, \|x\|$
    3. triangle inequality $\|x+y\| \leq \|x\| + \|y\| \quad \forall \; x, y \in X$

• <u>Ex:</u> given $X = \mathbb{C}^m$ then the l_p norms are defined:

$$\|x\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{1/p} \leftarrow \text{"p-norm"}, \quad p \geq 0$$

(i) if $p=0$, $\|x\|_0 = \#$ of entries of $x \neq 0$ = cardinality of the support of x

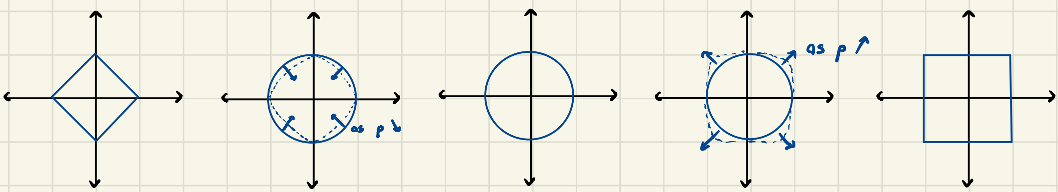
(ii) $p=1$, $\|x\|_1 = \sum_{i=1}^{m} |x_i| \leftarrow$ "taxi cab, manhattan distance"

(iii) $p=2$, $\|x\|_2 = \left(\sum_{i=1}^{m} |x_i|^2 \right)^{1/2} \leftarrow$ Euclidean distance

(iv) lim $p \to \infty$, $\|x\|_\infty = \max_i \{ |x_i| \}$

• to visualize the norms, often use the "unit ball" associated w/ those norms

• <u>Def:</u> the unit ball is $\{ x \text{ s.t. } \|x\| = 1 \}$

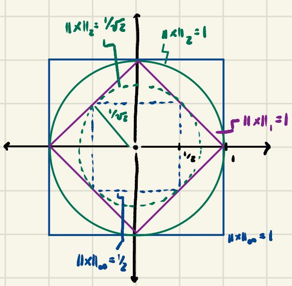


$p=1$ $\qquad$ $p<2$ $\qquad$ $p=2$ $\qquad$ $p>2$ $\qquad$ $p \to \infty$

• Norm Equivalency/Consistency: two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ are equivalent if $\exists\ c, C \in \mathbb{R}^+$, $c \leq C$ s.t.

$$c\|x\|_a \leq \|x\|_b \leq C\|x\|_a \quad \forall x.$$

• <u>Ex:</u> all p-norms w/ $p \geq 1$ are equivalent... let's consider $\|x\|_1$, $\|x\|_2$, $\|x\|_\infty$ for $x \in \mathbb{R}^2$



$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{2}\,\|x\|_2 \leq 2\,\|x\|_\infty$$

"contains" in

generically: if $x \in \mathbb{C}^m$, then ℓ_p & ℓ_q w/ $1 \le p \le q$

$$\|x\|_q \le \|x\|_p \le m^{(1/p - 1/q)} \|x\|_q$$

$$\left.\begin{aligned} \text{so: } \|x\|_2 &\le \|x\|_1 \le \sqrt{m}\, \|x\|_2 \\ \|x\|_\infty &\le \|x\|_2 \le \sqrt{m}\, \|x\|_\infty \\ \|x\|_\infty &\le \|x\|_1 \le m\, \|x\|_\infty \end{aligned}\right\} \|x\|_\infty \le \|x\|_2 \le \|x\|_1 \le \sqrt{m}\, \|x\|_2 \le m\, \|x\|_\infty$$

- further norm inequalities... follow from Holder's inequality ⇒ Cauchy-Schwartz
  Jensen's inequality when convex

- Matrix/Operator Norms: two perspectives

  1. "Entrywise": treats $A \in \mathbb{C}^{m \times n}$ as a list of mn #'s ... views A as some object
     - Ex: Frobenius norm $\|A\|_F = \left(\sum_{i,j} |a_{ij}|^2 \right)^{1/2}$

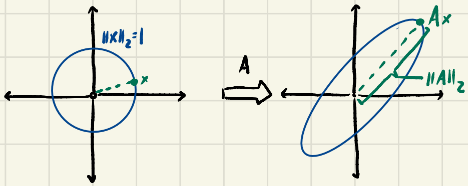
  2. "Induced norms": view $A \in \mathbb{C}^{m \times n}$ as parametrizing a transform $T(x): \mathbb{C}^n \to \mathbb{C}^m$, $T(x) = Ax$
     (Operator norm)

     - Def: given $\mathbb{C}^n$, $\mathbb{C}^m$ vector spaces w/ norms $\|\cdot\|_a$, $\|\cdot\|_b$ the induced operator norm of $A \in \mathbb{C}^{m \times n}$ is

$$\sup_{\substack{x \neq 0 \\ x \in \mathbb{C}^n}} \left\{ \frac{\|Ax\|_b}{\|x\|_a} \right\} = \sup_{\substack{\|x\|_a = 1 \\ x \in \mathbb{C}^n}} \left\{ \|Ax\|_b \right\}$$

       - size of maximum output (measured w/ $\|\cdot\|_b$) given input x s.t. $\|x\|_a = 1$
       - then: $\|Ax\|_b \le \|A\|_{(a,b)} \|x\|_a$

     - Def: induced p-norm of A is $\sup_{\substack{x \in \mathbb{C}^n \\ \|x\|_p = 1}} \left\{ \|Ax\|_p \right\}$ ← ℓ_p norm.
       - maximum "amplification" under multiplication by A...



nontrivial to compute in most cases

- operator norms inherit the norm equivalency of the vector norms that induce them
- allows comparison/inequalities relating operator norms
  - useful since some induced p-norms can be computed entrywise...

j^{th} column of A

- Ex: $\|A\|_1 = \max\limits_{1\le j\le n} \left\{ \sum\limits_{i=1}^{m} |a_{ij}| \right\} = \max\limits_{1\le j\le n} \left\{ \|\vec{a}_j\|_1 \right\} =$ "max col. sum"

$\|A\|_\infty = \max\limits_{1\le i\le m} \left\{ \sum\limits_{j=1}^{n} |a_{ij}| \right\} = \max\limits_{1\le i\le m} \left\{ \| - a_i - \|_1 \right\} =$ "max row sum"

- Why?
  - Proof: $\|A\|_1 = \sup\limits_{\|x\|_1 = 1} \left\{ \|Ax\|_1 \right\}$ $\quad \|x\| = \sum\limits_{j=1}^{n} |x_j| = 1$

$$\|Ax\|_1 = \Big\| \sum_{j=1}^{n} x_j \vec{a}_j \Big\|_1 \overset{\text{triangle}}{\le} \sum_{j=1}^{n} |x_j| \, \|\vec{a}_j\|_1$$

$\therefore$ $\|Ax\|_1$ is a weighted average of the 1-norm of the col's of A...

$\le \max\limits_{1\le j\le n} \left\{ \|\vec{a}_j\|_1 \right\}$ w/ equality if $x = e_M = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow M$

for M maximizing $\|a_j\|_1$ over all j

(x is indicator vector for col. of maximal sum)

  - $\|A\|_\infty = \sup\limits_{\|x\|_\infty = 1} \left\{ \|Ax\|_\infty \right\} = \ldots$ try and prove this yourself.

- Equivalency: given $A \in \mathbb{C}^{m\times n}$, $\frac{1}{\sqrt{n}}\|A\|_\infty \le \|A\|_2 \le \sqrt{m}\,\|A\|_\infty$, $\frac{1}{\sqrt{m}}\|A\|_1 \le \|A\|_2 \le \sqrt{n}\,\|A_1\|$

and, by Holder:

$$\|A\|_2 \le \sqrt{\|A\|_1 \|A\|_\infty}$$

  - useful since $\|A\|_2$ is hard to compute, but the most natural while $\|A\|_1$ & $\|A\|_\infty$ are easy to compute.

- Ex: recall $f(x) = x_1 - x_2$, $\hat{\kappa}_{a,b}(f,x) = \lim\limits_{\delta\to 0} \sup\limits_{\|\delta x\| \le \delta} \left\{ \frac{\|f(x+\delta x) - f(x)\|_b}{\|\delta x\|_a} \right\}$

if f is continuously differentiable, $f(x+\delta x) = f(x) + \underset{\uparrow\,\text{Jacobian}}{J(x)}\,\delta x + \mathcal{O}(\delta x^2)$ (local Taylor series)

for smooth enough norms

$$\text{so} \quad \lim_{\substack{\delta\to 0 \\ \|\delta x\|\le\delta}} \left\{ \frac{\|f(x+\delta x) - f(x)\|_b}{\|\delta x\|_a} \right\} = \lim_{\substack{\delta\to 0 \\ \|\delta x\|\le\delta}} \left\{ \frac{\|J(x)\,\delta x + \mathcal{O}(\delta x^2)\|_b}{\|\delta x\|_a} \right\} \doteq \sup_{\delta x \ne 0} \left\{ \frac{\|J(x)\,\delta x\|_b}{\|\delta x\|_a} \right\} = \|J(x)\|_{(a,b)}$$

we'd used $\|\cdot\|_{a=b=\infty}$ so wanted $\|J(x)\|_\infty = \|[1, -1]\|_\infty = 1 + 1 = 2$

if $\|\cdot\|_{a=b=1}$ ... $\|J(x)\|_1 = \max\{1, 1\} = 1$.

if $\|\cdot\|_{a=b=2}$ ... $n=2$, $m=1$ so $\frac{2}{\sqrt{2}} \le \|J(x)\|_2 \le 2$, $1 \overset{\text{norm equiv.}}{\le} \|J(x)\|_2 \le \sqrt{2}$, $\|J(x)\|_2 \overset{\text{Holder}}{\le} \sqrt{2}$

$\hat{\kappa}_\infty(x) = 2$, $\hat{\kappa}_1(x) = 1$, $\hat{\kappa}_2(x) \in [1, \sqrt{2}]$ — vary based on how we measure the size of the input x_1, x_2

- **Ex:** Conditioning of division: $f(x) = \frac{x_1}{x_2}$, then $J(x) = [\,1/x_2, \; -x_1/x_2^2\,]$

$$\left.\begin{aligned} &\text{so} \quad \hat{\kappa}_\infty(x) = \|J(x)\|_\infty = \frac{1}{|x_2|^2}(|x_1| + |x_2|) \\ &\|x\|_\infty = \max\{|x_1|, |x_2|\}, \quad \|f(x)\|_\infty = \frac{|x_1|}{|x_2|} \end{aligned}\right\} \quad \begin{aligned} \kappa_\infty(x) &= \hat{\kappa}_\infty(x) \frac{\|x\|_\infty}{\|f(x)\|_\infty} = \frac{|x_1| + |x_2|}{|x_2|^2} \frac{|x_2|}{|x_1|} \|x\|_\infty \\ &= \frac{|x_1| + |x_2|}{|x_1||x_2|} \max\{|x_1|, |x_2|\} \\ &= 1 + \max\left\{\frac{|x_1|}{|x_2|}, \frac{|x_2|}{|x_1|}\right\} \end{aligned}$$

so $\kappa_\infty(x) = 1 + \max\left\{\frac{|x_1|}{|x_2|}, \frac{|x_2|}{|x_1|}\right\}$ ... division is ill-conditioned if $|x_1| \gg |x_2|$ or $|x_2| \gg |x_1|$ (inputs of different scales)

- numerical moral: only trust division of #'s whose relative orders of magnitude are not overly different...
  expect to lose as many digits of accuracy as the relative orders of magnitudes of the inputs
  i.e. avoid dividing very large #'s by very small #'s.

Thursday - 03/30/2023

- **Logistics:**
  - HW 1 posted, due next Tuesday
  - First lab session on Friday, 3:00 - 5:00 pm, Jones 226
  - Resources: reading on discretization $ differencing posted

- **Goals:**
  - **Linear Transformations:**
    - Vector Spaces $ Transforms
    - Discretization example: differencing $ convolution
    - Matrix Products (performing linear transforms):
      - Cost/Complexity
      - Spectral Perspective
      - Conditioning

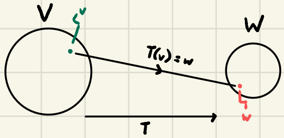
Linear Transformations:

- **Def:** a vector space V is a set of objects $v \in V$ called vectors equipped w/ vector addition, $v + w$ $ scalar multiplication, αv (over a field) that is closed under linear combination:

$$\text{given } v, w \in V, \text{ scalars } \alpha, \beta: \quad \underbrace{\alpha v + \beta w}_{\text{linear combination}} \in V$$

- **Def:** an operation/transform $T: V \to W$ (that maps between vector spaces V $ W) is linear if $\forall\ u, v \in V,\ \alpha, \beta$:

$$T(\alpha u + \beta v) = \alpha\, T(u) + \beta\, T(v)$$



- **Ex:** 1. $X = \mathbb{C}^n$ = all lists of n complex #'s, $A \in \mathbb{C}^{m \times n}$

  then let $T(x) = Ax$ for $x \in \mathbb{C}^n$

$$T: \underset{\substack{\uparrow \\ X}}{\mathbb{C}^n} \to \underset{\substack{\uparrow \\ Y}}{\mathbb{C}^m}$$

  and: $T(\alpha u + \beta v) = A(\alpha u + \beta v) = \alpha A u + \beta A v = \alpha\, T(u) + \beta\, T(v)$

- **Fact:** All linear transforms $T: X \to Y$ between finite dimensional vector spaces can be expressed $T(x) = Ax$ for some matrix $A \in \mathbb{C}^{m \times n}$
  - **Consequence:** approximate linear transformations numerically via multiplication w/ a matrix A.
    - view matrices / matrix products as discretized transforms

- Let's look at some other important/interesting vector spaces & associated transforms...

  - **Ex:** the set of all $m \times n$ matrices, $\mathbb{C}^{m \times n}$ is a vector space
    given $A \in \mathbb{C}^{m \times n}$ all transforms $T(A) = BAC$ for $B \in \mathbb{C}^{p \times m}$, $C \in \mathbb{C}^{n \times q}$ is a linear transformation...

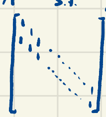
    Check: $B(\alpha A + \beta M)C = \alpha BAC + \beta BMC$
    $T(\alpha A + \beta M) = \alpha T(A) + \beta T(M)$. ✓

  - **Ex:** given $\Omega = \mathbb{C}$, $P^{(m)} = \{$all polynomials of a single variable of degree $\leq m\}$
    then $p \in P^{(m)}$ if $\exists$ coefficients $\alpha(p)$ s.t. $p(x) = \sum_{j=0}^{m} \alpha_j x^j$
    - is this a vector space $\forall m$? $\alpha p(x) + \beta q(x) =$ polynomial of deg. m ✓
      - what is it's dimension? $m+1$
    - is the following linear? check closed under lin. comb. ✓
      $$T[p](x) = q(x)\,p(x) \quad \text{for } q \in P^{(n)}$$
    - what is the space of outputs? polynomials of degree $m+n$, $P^{(n+m)}$
    - suppose $q(x) = x+1$. What is the matrix implementation of the transform?

      - $$T[p] = (x+1)\,p(x|\alpha) = \sum_{j=0}^{m} (x+1)\,\alpha_j x^j = \underbrace{\sum_{j=0}^{m} \alpha_j x^j}_{p(x|\alpha)} + \underbrace{\sum_{j=0}^{m} \alpha_j x^{j+1}}_{\text{degree } m+1 \text{ polynomial}} = \sum_{j=0}^{m+1} \beta_j x^j$$

$$\beta_j(\alpha) = \begin{cases} \beta_0 & : \alpha_0 \\ \beta_j \text{ if } 1 \leq j \leq m & : \alpha_j + \alpha_{j-1} \\ \beta_{m+1} & : \alpha_m \end{cases}$$ the output β_j is a lin. comb. of the α's

$\exists$ a matrix A s.t. $\beta = A\alpha$. A is $(m+2) \times (m+1)$

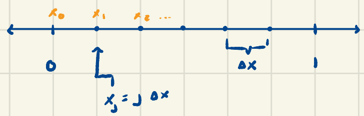
$A =$  ← notice: A is almost entirely 0... A is sparse

- **Ex:** given the domain $\Omega = [0,1]$ the set of all continuously differentiable, boundable functions $f: \Omega \to \mathbb{R}$ is a vector space and:

  1. $T(f) = \frac{d}{dx} f$ is a linear transform, $\frac{d}{dx}(\alpha f + \beta g) = \alpha \frac{d}{dx} f + \beta \frac{d}{dx} g$
  2. $T(f) = \int f$ is a linear transform, ...
  3. $T(f) = \sum_{j=-l}^{p} \alpha_j \left(\frac{d}{dx}\right)^j f$ is a linear transform, ...

- can we approximate these w/ a finite dimensional discretization?
  - what errors do discretization introduce?

- let's try to approximate $\frac{d}{dx}$, $f: [0,1] \to \mathbb{R}$, we'll assume that f is periodic w/ period 1
  - let's discretize the domain $\Omega = [0,1]$



$$\{x_j\}_{j=0}^{n-1}, \quad \Delta x = 1/n$$

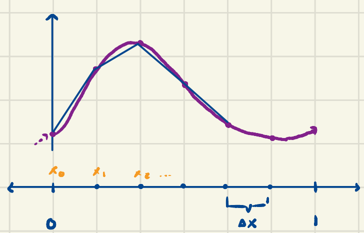
- replace functions over $\Omega \to$ samples: $\{f_j\}_{j=0}^{n-1}$, $f_j = f(x_j)$ exact sampling of $\frac{d}{dx}$

  samples: $\{f'_j\}_{j=0}^{n-1}$, $f'_j = \frac{d}{dx} f(x)\big|_{x_j}$

approx: $\underset{\substack{\uparrow \\ \text{converges as}}}{\hat{f}' \approx f'}$, $\hat{f}' = D(\Delta x) f$ $\Delta x \to 0$

- how to build $D(\Delta x)$? $\frac{d}{dx} f(x_j) = \lim_{h\to 0} \frac{f(x_j+h) - f(x_j)}{h} \approx \frac{f(x_j+\Delta x) - f(x_j)}{\Delta x}$

$$= \frac{f_{j+1} - f_j}{\Delta x} = \left(\tfrac{1}{\Delta x}\right)(\underset{\substack{\uparrow \\ \text{mod}(n)}}{f_{j+1}} - f_j)$$



$$\hat{f}'_j \approx \tfrac{1}{\Delta x}(f_{j+1} - f_j) = D(\Delta x) f$$

$$D(\Delta x) = \frac{1}{\Delta x}\underbrace{\begin{bmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & -1 & 1 & \\ & & & \ddots & \ddots \\ 1 & & & & -1 \end{bmatrix}}_{\text{circulant}}$$

($n \times n$) super-sparse: $2n$ nonzero entries, n^2 total entries

- **Ex:** given $\Omega \subset \mathbb{R}^d$, compact, the set of boundable functions $f: \Omega \to \mathbb{R}$ is a vector space and, given $K: \Omega \times \Omega \to \mathbb{R}$, $|K(x,y)| < \infty$ a.e.

$$\underbrace{T(f) = \int_{y\in\Omega} K(x,y) f(y)\, dy}_{\text{integral transform}} \quad \text{is a linear transform}$$

(very general, includes all convolutions, Fourier & Laplace transforms, etc.)

oh, ... and in limits all differential & integral operations.

- the functional analog to matrix vector products...

- Error analysis: $f(x_j + \Delta x) - f(x_j)$

using Taylor series: $f(x_j+\Delta x) = f(x_j) + \big(\underbrace{\tfrac{d}{dx} f(x_j)}_{f'(x_j)}\big)\Delta x + \tfrac{1}{2}\left(\tfrac{d^2}{dx^2} f(x_j)\right)\Delta x^2 + \mathcal{O}(\Delta x^3)$

$$\tfrac{1}{\Delta x}\big(f(x_j+\Delta x) - f(x_j)\big) = \tfrac{1}{\Delta x}\left(f'(x_j)\Delta x + \tfrac{1}{2}\left(\tfrac{d^2}{dx^2} f(x_j)\right)\Delta x^2 + \mathcal{O}(\Delta x^3)\right)$$

$$= f'(x_j) + \tfrac{1}{2}\left(\tfrac{d}{dx}\right)^2 f(x_j)\,\Delta x + \mathcal{O}(\Delta x^2)$$

• Matrix Products:

1. inner product: product of two vectors $u, v \in V$ (notation: $\langle \overset{u}{\cdot}, \overset{v}{\cdot} \rangle$ or $\underline{u} \cdot \underline{v}$ or $u^T v$)

$$\langle \cdot, \cdot \rangle : V \times V \to \mathbb{C} \text{ s.t.}$$

   1. bilinear: $\langle u + v, w \rangle = \langle u, w \rangle + \langle v, w \rangle$

      and $\langle u, v + w \rangle = \langle u, v \rangle + \langle u, w \rangle$

   2. $\langle \alpha u, v \rangle = \alpha \langle u, v \rangle = \langle u, \alpha v \rangle$

   3. commutative: $\langle u, v \rangle = \langle v, u \rangle$

   4. $\langle v, v \rangle \geq 0$ and $= 0$ iff $v = 0$ ← induces a norm: $\|v\|^2 = \langle v, v \rangle$

conjugate transpose

if V is finite dimensional then $\langle v, u \rangle = v^* M u$

for $v, u \in \mathbb{C}^n$, $M \in \mathbb{C}^{n \times n}$ p.d., and $v^* = [\bar{v}_1, \bar{v}_2, \dots \bar{v}_n]$

usually:

$$\langle v, u \rangle = v^* u = \underbrace{\sum_{j=1}^{n} \bar{v}_j u_j}_{\text{sum of elementwise product}} \longleftarrow \text{row} \times \text{col.} \quad [\bar{v}_1, \bar{v}_2, \dots \bar{v}_n] \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

2. Matrix-vector product: given $A \in \mathbb{C}^{m \times n}$, $x \in \mathbb{C}^n$ then $b = Ax$ where

   (i) Elementwise: $b_i = [Ax]_i = \sum_{j=1}^{n} a_{ij} x_j$

   (ii) Row-wise: view A as a collection of rows, ← $A = \begin{bmatrix} — a_1 — \\ — a_2 — \\ \vdots \\ — a_m — \end{bmatrix}$ (width n) each row is a vector in $\mathbb{C}^n$, the input space

   Ax as a series of inner products:

$$b_i = [Ax]_i = (— \, i^{th} \text{ row of } \bar{A} —) \cdot x$$

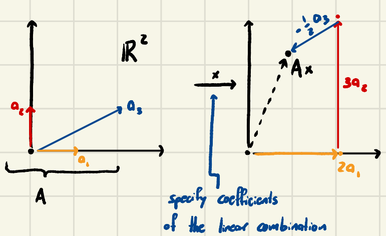
   (iii) Column-wise: view A as a collection of columns, ← $A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix}$ (height m) each col. is a vector in $\mathbb{C}^m$, the output space

   Ax as a linear comb. of the col's of A

$$Ax = \sum_{j=1}^{n} x_j \, a_j$$

coeff. of this combination

• Ex: $A = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix}$, $x = \begin{bmatrix} 2 \\ 3 \\ -\frac{1}{2} \end{bmatrix}$,



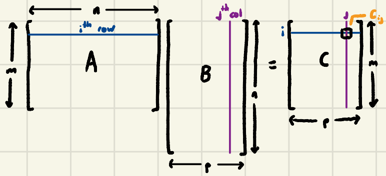
3. <u>Matrix-matrix products</u>: given $A \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{n \times p}$ then $C = AB \in \mathbb{C}^{m \times p}$ where:

(i) <u>Element-wise</u>: view A as a collection of rows, B of col's

then:

j^{th} column of B

$$C_{ij} = [AB]_{ij} = (\text{— } i^{th} \text{ row of } \bar{A} \text{ —}) \cdot b_j = \sum_{k=1}^{n} a_{ik} b_{kj}$$



(ii) <u>Column-wise</u>: $AB = A[b_1 \cdots b_p] = [Ab_1 \cdots Ab_p]$ (iii) row-wise...

- implies the <u>outer product</u>: $u \in \mathbb{C}^n$, $v \in \mathbb{C}^m$ then $u \in \mathbb{C}^{m \times 1}$ (col.), $v^* \in \mathbb{C}^{1 \times n}$ (row)

$$uv^* = m\left[\begin{matrix} u_1 \\ \vdots \\ u_m \end{matrix}\right]\overbrace{\left[\bar{v}_1 \cdots \bar{v}_n\right]}^{n} = \overbrace{\begin{bmatrix} u_1\bar{v}_1 & u_1\bar{v}_2 & \cdots & u_1\bar{v}_n \\ u_2\bar{v}_1 & u_2\bar{v}_2 & \cdots & u_2\bar{v}_n \\ \vdots & \vdots & & \vdots \\ u_m\bar{v}_1 & u_m\bar{v}_2 & \cdots & u_m\bar{v}_n \end{bmatrix}}^{n} m$$

(iv) Can also compute AB via a sum of outer products...

$$AB = \sum_{k=1}^{n} (k^{th} \text{ col. of } A)(k^{th} \text{ row of } B)$$

- <u>Computational Cost/Complexity of Matrix Products</u>: how expensive are these operations?

1. inner products of $x, y \in \mathbb{C}^n \rightarrow x^* y = \sum_{j=1}^{n} \bar{x}_j y_j \Rightarrow \left.\begin{matrix} n \text{ products} \\ n-1 \text{ addition} \end{matrix}\right\} 2n-1 = \mathcal{O}(n)$

2. matrix-vector: $A \in \mathbb{C}^{m \times n}$, $x \in \mathbb{C}^n \longrightarrow Ax = m$ inner prod. of vectors of size n
$= \mathcal{O}(mn)$

3. matrix-matrix: $A \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{n \times p} \rightarrow AB = mp$ inner prod. ...
$= \mathcal{O}(mnp)$

- for ex: $A \in \mathbb{C}^{n \times n}$, $B \in \mathbb{C}^{n \times n} \rightarrow$ comp. AB, cost $\mathcal{O}(n^3)$
matrix-matrix mult. has a <u>cubic cost</u> (in flops)

$\exists$ optimized or faster methods:

schoolbook: $\mathcal{O}(n^3)$, Strassen: $\mathcal{O}(n^{2.807})$, Coppersmith-Winograd $\mathcal{O}(n^{2.3})$

- Can also speed computation by exploiting structure in our matrices:

  1. if A is sparse (# of nonzero entries of $A = |\text{supp}(A)| \ll mn$)
     use sparse operations, cost $Ax = \mathcal{O}(|\text{supp}(A)|)$
     - let $\text{supp}(A) = \{i, j \text{ s.t. } a_{ij} \neq 0\}$
     - then Ax only requires 1 mult. & 1 addition per $a_{ij} \neq 0$
       so cost using sparse operations is $\mathcal{O}(|\text{supp}(A)|)$
       - <u>Ex:</u> $D(\Delta x)$ has $|\text{supp}| = 2n$, size n^2, cost of $D(\Delta x)f$ is $\mathcal{O}(n^2)$ w/out exploiting sparsity, $\mathcal{O}(n)$ w/ sparsity.

  2. symmetries of A to speed comp.
     - <u>Ex:</u> if A implements a DFT on n samples
       use multiscale sym. of A to perform $Ax = \mathcal{O}(n \ln(n))$
     - other examples: fast Hadamard, fast Hankel (constant down diagonals)

- <u>Stability & Conditioning of Linear Transforms</u>: usually requires a different perspective on $T(x) = Ax \ldots$

  - <u>Spectral Perspective</u>:
    - matrices A can be factored into "simpler" factors: $A = F_1 F_2 \ldots F_\ell = \prod_{j=1}^{\ell} F_j$
      where the factors F_j each perform a "simple" transform.
    - Key factorizations:
      1. if $A \in \mathbb{C}^{n \times n}$ is diagonalizable ($\exists$ n lin. ind. vectors $v \in \{v_j\}_{j=1}^{n}$ s.t. $Av_j = \lambda_j v_j$ for some scalar $\lambda_j \in \mathbb{C}$)

         then: $A = V \Lambda V^{-1}$

         where: $V = \underbrace{\begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix}}_{\text{eigenvectors}}$, $\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} = \text{diag}(\underbrace{\lambda_1, \lambda_2, \ldots \lambda_n}_{\text{eigenvalues}})$

      2. <u>Singular Value Decomp (SVD)</u>: given any $A \in \mathbb{C}^{m \times n}$

         then: $A = U \Sigma V^*$

         where: ?
         ?
         ?

$$U = \underbrace{\overbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{bmatrix}}^{m}}_{\text{left singular vectors}} \Big\updownarrow m, \quad \Sigma = m \Big\updownarrow \overbrace{\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \; 0 \\ \hline & & & \end{bmatrix}}^{n}_{\text{singular values}}, \quad V = \underbrace{\overbrace{\begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix}}^{n}}_{\text{right singular vectors}} \Big\updownarrow n$$

- Eigenvalue Example: if $A \in \mathbb{C}^{n \times n}$ w/ n linearly independent eigenvectors v

  then $\forall\ x \in \mathbb{C}^n\ \exists\ y \in \mathbb{C}^n$ s.t. $x = \sum_{j=1}^{n} y_j v_j = Vy \Rightarrow y = V^{-1}x$

  in eigenbasis, $A \sim \Lambda$, just scales by λ's.

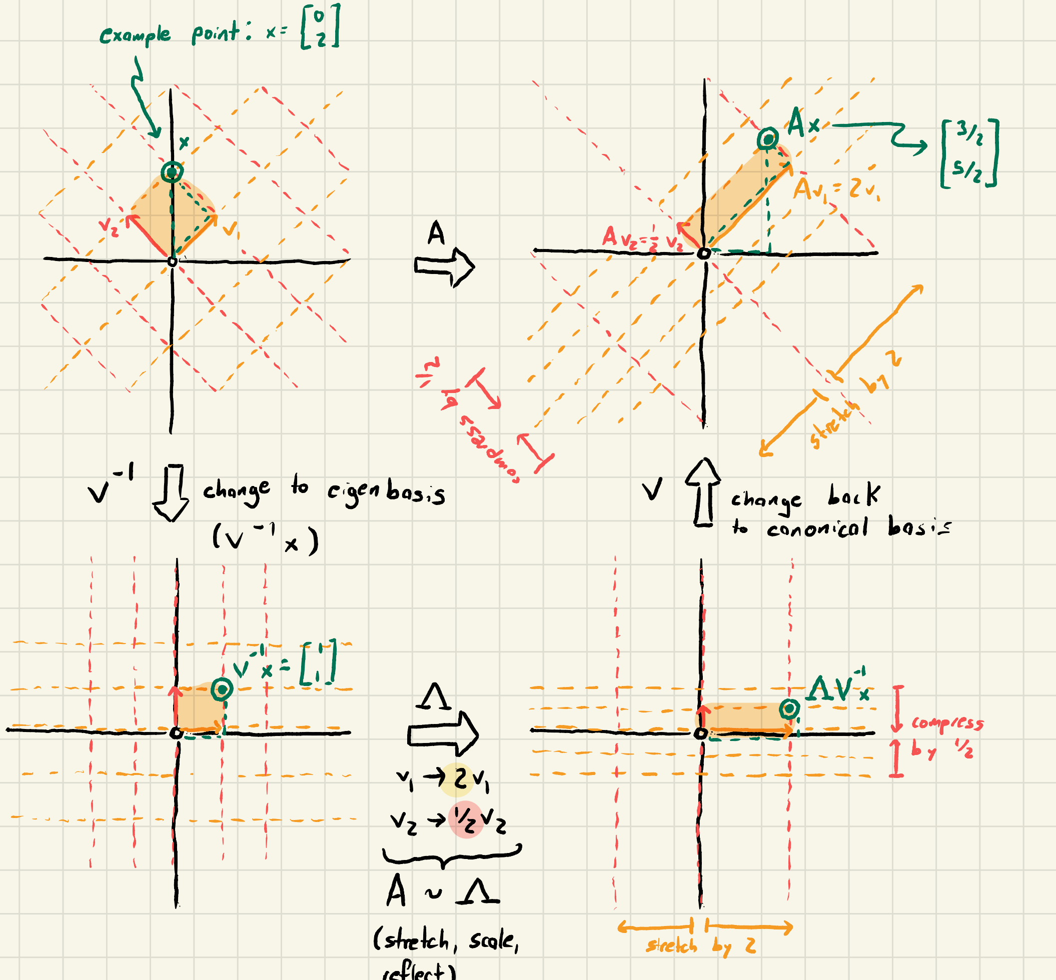
  $$\therefore\ Ax = V\Lambda V^{-1}x = V\Lambda y = \sum_{j=1}^{n} (\Lambda y)_j v_j = \sum_{j=1}^{n} (\lambda_j y_j) v_j$$

- Suppose: $A = \begin{bmatrix} 5/4 & 3/4 \\ 3/4 & 5/4 \end{bmatrix}$

  then $\lambda = 2, 1/2$ so $\Lambda = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$

  and $V = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$

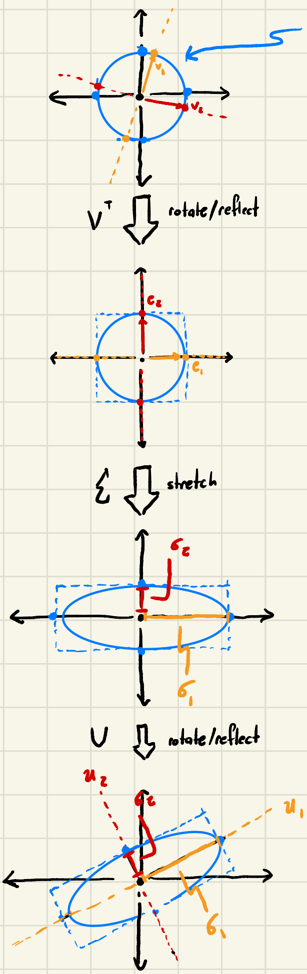
  $V^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ exists so $\rightarrow A = V\Lambda V^{-1}$



- **SVD**: $A = U\Sigma V^*$ (where U, V have $\perp$ normalized columns, Σ diagonal, real, nonnegative, nonincreasing)

  takes one orthonormal basis, $V = \{v_1, v_2, \dots v_n\}$, and maps it to an orthogonal basis, $U\Sigma = \{\sigma_1 u_1, \sigma_2 u_2, \dots \sigma_n u_n\}$

  - can help to visualize the transform of the unit ball



unit ball: $\{$all x s.t. $\|x\| = 1\}$

1. rotation, but rotation doesn't change the unit ball (ignore V^T)

2. multiply by Σ, scale the directions $\begin{bmatrix}1\\0\end{bmatrix}$ by σ_1, $\begin{bmatrix}0\\1\end{bmatrix}$ by σ_2

   turns the unit ball into an ellipse with axes lengths equal to the singular values

3. rotation, this one matters b/c the ellipse is <u>not</u> rotationally

   - need: $Ue_1 = U\begin{bmatrix}1\\0\end{bmatrix} = u_1$

     $Ue_2 = U\begin{bmatrix}0\\1\end{bmatrix} = u_2$

   that means that the columns of U, $u_1, u_2, \dots$ are the directions of the principal axes of the ellipse.

- so, A sends right singular vectors $\underset{\text{orthonormal basis for } \mathbb{R}^n}{\{v_j\}_{j=1}^n}$ to scaled left singular vectors $\underset{\text{orthogonal basis for range}(A) \subseteq \mathbb{R}^m}{\{\sigma_j u_j\}_{j=1}}$

- what about A^T? $A = U\Sigma V^T$

  $A^T = \underbrace{V\Sigma^T U^T}$ so V plays the role of U for A^T

  exchange right & left singular vectors, singular values unchanged

  - the singular vectors $v_1, v_2, \dots$ orient the ellipse associated with A^T
  - A^T sends $\{u_j\}_{j=1}^m$ to $\{\sigma_j v_j\}_{j=1}$

Week 3 - Linear Transforms & Orthogonalization

Tuesday - 4/04/2023

- **Logistics:**
  - HW1 due tonight
  - Reading: finish Chapter II, ~~ ~~ except lecture 11, lecture 16
  - HW 2 assigned, due next Tuesday
  - Project 1 will post Wednesday
- **Goals:**
  - Linear Transforms:
    - The spectral perspective
    - Conditioning & Stability

Linear Transforms continued...

- **Spectral Perspective:**
  - matrices A can be factored into "simpler" factors: $A = F_1 F_2 \dots F_\ell = \prod_{j=1}^{\ell} F_j$
    where the factors F_j each perform a "simple" transform.
  - Key factorizations:

    1. if $A \in \mathbb{C}^{n \times n}$ is diagonalizable ($\exists$ n lin. ind. vectors $v \in \{v_j\}_{j=1}^{n}$ s.t. $Av_j = \lambda_j v_j$ for some scalar $\lambda_j \in \mathbb{C}$)

       then: $A = V \Lambda V^{-1}$

       where: $V = \underbrace{\begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix}}_{\text{eigenvectors}}$, $\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} = \mathrm{diag}(\underbrace{\lambda_1, \lambda_2, \dots \lambda_n}_{\text{eigenvalues}})$

    2. Singular Value Decomp (SVD): given any $A \in \mathbb{C}^{m \times n}$

       then: $A = U \Sigma V^*$

       where: U is $m \times m$ & has $\perp$, normalized col's

       V is $n \times n$ & ...

       Σ is diagonal, $\sigma_j \in \mathbb{R}^+$, $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \dots \sigma_{\mathrm{rank}(A)} \geq 0$
       $(m \times n)$

       relation to eigenvalues/vectors
       - $\sigma_j^2 = \lambda_j(A^*A) = \lambda_j(AA^*)$ (up to rank)
       - u_j = eigenvectors of AA^*
       - v_j = eigenvectors of A^*A.

$$U = \underbrace{\overset{\longleftarrow m \longrightarrow}{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{bmatrix}}}_{\text{left singular vectors}} \updownarrow m, \quad \Sigma = m \updownarrow \overset{\longleftarrow n \longrightarrow}{\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \\ & & & 0 \end{bmatrix}}_{\text{singular values}}, \quad V = \underbrace{\overset{\longleftarrow n \longrightarrow}{\begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix}}}_{\text{right singular vectors}} \updownarrow n$$

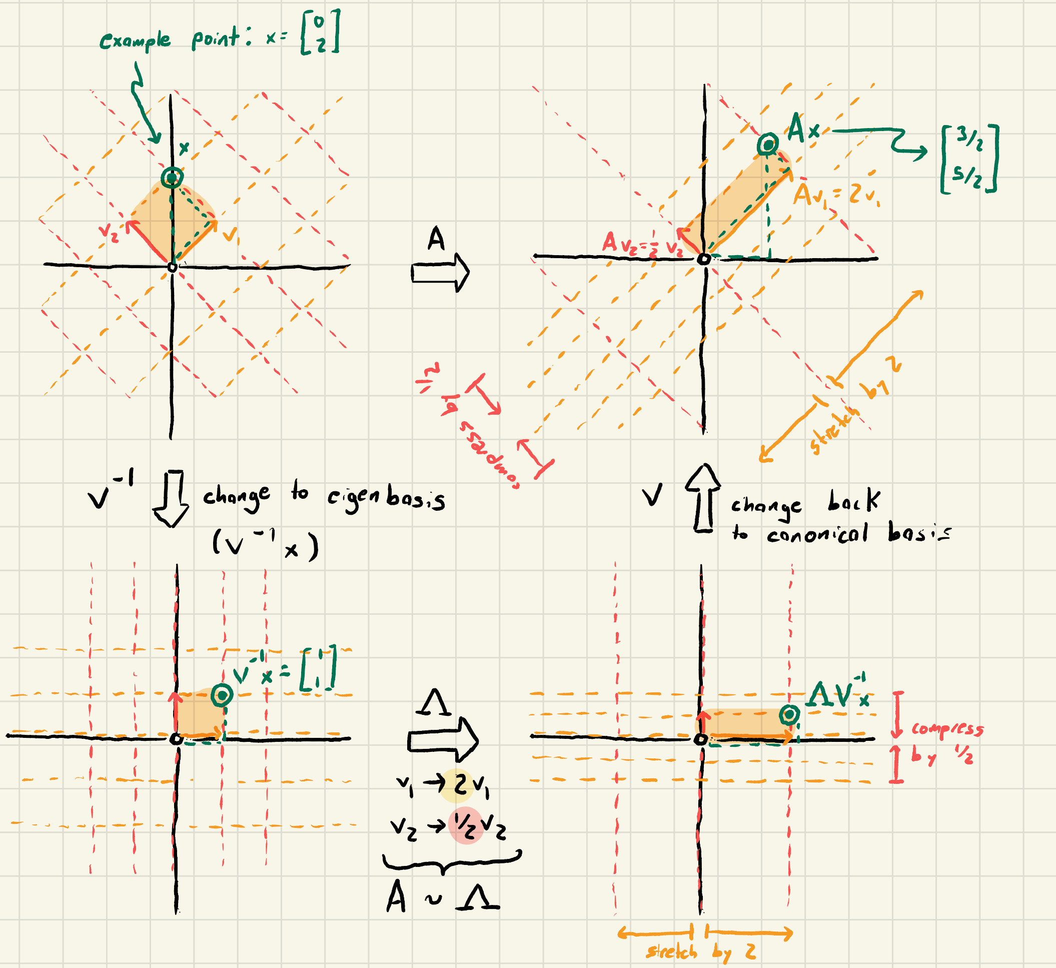
- **Eigenvalue Example**: if $A \in \mathbb{C}^{n\times n}$ w/ n linearly independent eigenvectors v
  then $\forall\ x \in \mathbb{C}^n\ \exists\ y \in \mathbb{C}^n$ s.t. $x = \sum_{j=1}^{n} y_j v_j = Vy \Rightarrow y = V^{-1}x$

  $$\therefore\ Ax = V\Lambda V^{-1}x = V\Lambda y = \sum_{j=1}^{n} (\Lambda y)_j v_j = \sum_{j=1}^{n} \underbrace{(\lambda_j y_j)}_{\text{in eigenbasis, } A \sim \Lambda \text{, just scales by } \lambda\text{'s.}} v_j$$

- Suppose: $A = \begin{bmatrix} 5/4 & 3/4 \\ 3/4 & 5/4 \end{bmatrix}$ then $\lambda = 2, 1/2$ so $\Lambda = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$

  and $V = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$

  $V^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ exists so $\rightarrow A = V\Lambda V^{-1}$

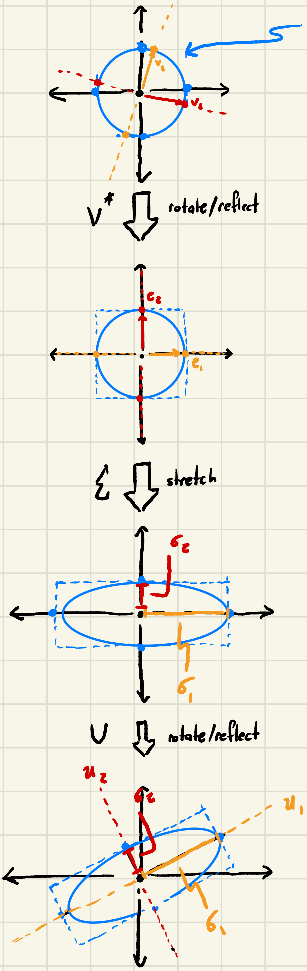


- **SVD**: $A = U\Sigma V^*$ (where U, V have $\perp$ normalized columns, Σ diagonal, real, nonnegative, nonincreasing)

  takes one orthonormal basis, $V = \{v_1, v_2, \dots v_n\}$, and maps it to an orthogonal basis, $U\Sigma = \{\sigma_1 u_1, \sigma_2 u_2, \dots \sigma_n u_n\}$

  - can help to visualize the transform of the unit ball

unit ball: $\{$all x s.t. $\|x\| = 1\}$



1. rotation, but rotation doesn't change the unit ball (ignore V^T)

2. multiply by Σ, scale the directions $\begin{bmatrix}1\\0\end{bmatrix}$ by σ_1, $\begin{bmatrix}0\\1\end{bmatrix}$ by σ_2

   turns the unit ball into an ellipse with axes lengths equal to the singular values

3. rotation, this one matters b/c the ellipse is <u>not</u> rotationally

   - need: $Ue_1 = U\begin{bmatrix}1\\0\end{bmatrix} = u_1$

     $Ue_2 = U\begin{bmatrix}0\\1\end{bmatrix} = u_2$

   that means that the columns of U, $u_1, u_2, \dots$ are the directions of the principal axes of the ellipse.

orthonormal basis for $\mathbb{R}^n$ orthogonal basis for range$(A) \subseteq \mathbb{R}^m$

- so, A sends right singular vectors $\{v_j\}_{j=1}^n$ to scaled left singular vectors $\{\sigma_j u_j\}_{j=1}$

- what about A^*? $A = U\Sigma V^*$

  $A^* = \underbrace{V\Sigma^T U^*}$ so V plays the role of U for A^T

  exchange right & left singular vectors, singular values unchanged

  - the singular vectors $v_1, v_2, \dots$ orient the ellipse associated with A^T

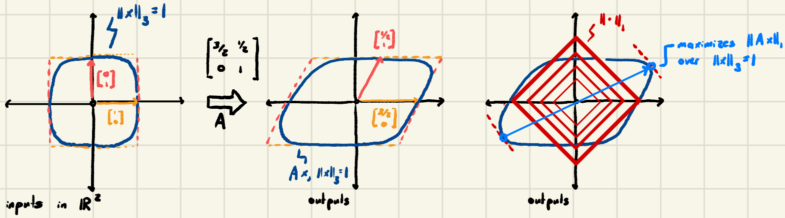
  - A^* sends $\{u_j\}_{j=1}^m$ to $\{\sigma_j v_j\}_{j=1}$ - A^{-1} sends $\{u_j\}_{j=1}^n$ to $\{\frac{1}{\sigma_j} v_j\}_{j=1}^n$

- The spectral perspective helps understand transforms, their induced norms, & their conditioning

  - Recall, given $A \in \mathbb{C}^{m \times n}$, and a pair of vector norms $\underbrace{\|\cdot\|_a}_{\text{inputs}}, \underbrace{\|\cdot\|_b}_{\text{outputs}}$ the induced a,b norm of A is

$$\|A\|_{a,b} = \sup_{\substack{x \in \mathbb{C}^n \\ x \neq 0}} \left\{ \frac{\|Ax\|_b}{\|x\|_a} \right\} = \sup_{\|x\|_a = 1} \left\{ \|Ax\|_b \right\}$$

  - Ex: $\|A\|_{3,1} = \sup_{\|x\|_3 = 1} \left\{ \|Ax\|_1 \right\}$



  - generically a hard optimization problem (Ex: even for $a=b=2$, Euclidean distance), often seek bounds

    - bounds on the induced norms arise naturally from the spectrum.

- Def: given $A \in \mathbb{C}^{m \times m}$, the spectral radius of A, $\rho(A) = \max_j \{ |\lambda_j(A)| \}$ (magnitude of the largest eigenvalue)
  - then:

$$\rho(A) \le \|A\| \text{ for any induced norm...}$$

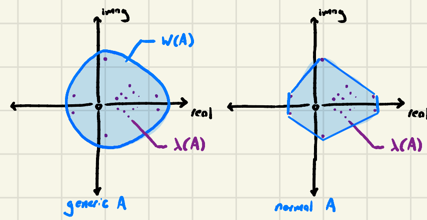
- More generally, consider the numerical range of A: $W(A) = \underset{x \neq 0}{\text{range}}(R_A(x))$ where $\underbrace{R_A(x)}_{\text{"Rayleigh Quotient"}} = \frac{x^* A x}{x^* x}$ (square A)

  - notice, if v is an eigenvector of A, $Av = \lambda v$ then $R_A(v) = \frac{v^* A v}{v^* v} = \frac{v^* \lambda v}{v^* v} = \lambda$

  - the numerical range is a compact (closed and bounded), convex subset of $\mathbb{C}$ which contains all eigenvalues of A

  - if A is:

$$\underbrace{\text{normal: } A^* A = A A^*}_{\text{commutes w/ its conjugate transpose}} \iff \underbrace{\text{unitarily diagonalizable}}_{n \perp \text{ eigenvectors}}$$



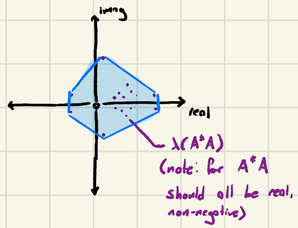
then W(A) is the convex hull of the eigenvectors

- especially useful for studying $\|A\|_2$ (induced by $\|x\|_2 = \sqrt{x^*x}$)

- **Def:** $\|A\|_2 = \sup_{\|x\|_2=1} \{ \|Ax\|_2 \}$ = largest ratio of output length to input length.

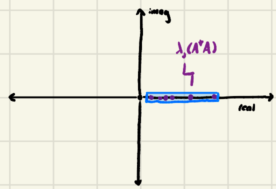
then: $\|A\|_2^2 = \sup_{x\neq 0} \left\{ \frac{\|Ax\|_2^2}{\|x\|_2^2} \right\} = \sup_{x \neq 0} \left\{ \frac{x^*A^*Ax}{x^*x} \right\} = \sup_{x\neq 0} \{ R_{A^*A}(x) \} = \sup \{ |w| \mid w \in W(A^*A) \}$

- Fact: A^*A is square for all $A \in \mathbb{C}^{m \times n}$ ($A^*A \in \mathbb{C}^{n\times n}$)
  and is normal ($(A^*A)^* = A^*A^{**} = A^*A$ $\therefore$ $(A^*A)^*(A^*A) = (A^*A)(A^*A)^*$)
  thus unitarily diagonalizable...



$\therefore$ $W(A^*A)$ = convex hull of $\{ \lambda_j(A^*A) \}$
= convex hull of $\{ \sigma_j^2(A) \}$

so, $\sup\{ |w| \mid w \in W(A^*A) \} = \max_j \{ \sigma_j^2(A) \}$

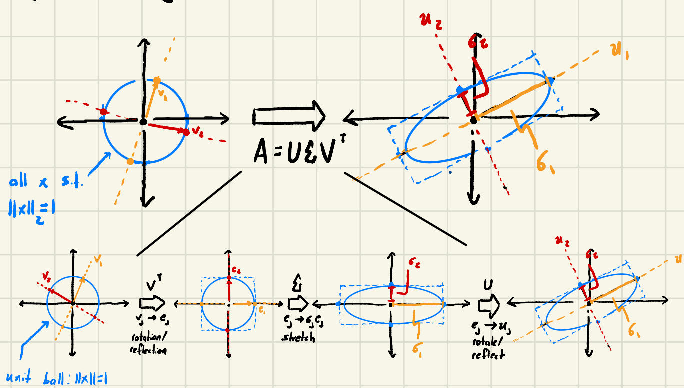


so, $\|A\|_2^2 = \max_j \{ \sigma_j^2(A) \} = \sigma_1^2(A)$

$\|A\|_2 = \sigma_1(A)$

moral: the induced 2-norm of a matrix is its largest (first) singular value.

- why? recall the geometry of the SVD



the length of the longest principal axis is σ_1 (input is v_1)

- Allows us to easily compute: $\|A^*\|_2 = \sigma_{max}(A) = \|A\|_2$
  (A^* takes $u_j \to \sigma_j v_j$)
  $\|A^{-1}\|_2 = 1/\sigma_{min}(A)$
  (A^{-1} takes $u_j \to \frac{1}{\sigma_j} v_j$)

- <u>Stability of Linear Transforms</u>:
    - linear transformations (matrix products) are stable (forward stable, i.e. $f(A,B) = AB$ then $\tilde{f}(A,B) = AB + \delta f(A,B)$ where $\underset{\uparrow \text{ entrywise}}{\dfrac{|\delta f|}{|A||B|}} = \mathcal{O}(\varepsilon_m)$)
    - inner products \$ matrix-vector products are stable \$ backward stable (if $f(A,x) = Ax$, $\tilde{f}(A,x) = Ax + \delta f(A,x)$, $|\delta f|/|A||x| = \mathcal{O}(\varepsilon_m)$ and $\tilde{f}(A,x) = f(\tilde{A},x)$ where $\tilde{A} = A + \delta A$, $\dfrac{|\delta A|}{|A|} = \mathcal{O}(\varepsilon_m)$

see resources on canvas

    - outer products are stable but not backward stable

    - generally: matrix products are stable \$ backward stable if reduce dimension (input → output)
      not backward stable if increase dimension.

    - the degree of stability (constant in $\mathcal{O}(\varepsilon_m)$ statements) depends on the conditioning of the product...
        - <u>Ex</u>: compute $f(A,x) = Ax$
          then $\dfrac{\|\tilde{f}(A,x) - f(A,x)\|}{\|f(A,x)\|} = \mathcal{O}(\underset{\substack{\uparrow \\ \text{condition \# of } A}}{\kappa(A)}\, \varepsilon_m)$

- <u>Conditioning of linear transforms</u>: given $A \in \mathbb{C}^{m \times n}$, how ill/well conditioned are products w/ A?
    - given $A \in \mathbb{C}^{m\times n}$, $x \in \mathbb{C}^n$, $\overbrace{f(x) = Ax}^{\text{problem is, perform the transformation}}$, and some norm $\|\cdot\|$ on $\mathbb{C}^m$ \$ $\mathbb{C}^n$
      then:

$$\kappa(A,x) = \lim_{\delta \to 0} \sup_{\|\delta x\| \le \delta} \left\{ \frac{\|A(x+\delta x) - Ax\|}{\|Ax\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right\} = \overbrace{\underbrace{\lim_{\delta \to 0} \sup_{\|\delta x\| \le \delta} \left\{ \frac{\|A\delta x\|}{\|\delta x\|} \right\}}_{\hat{\kappa}(A,x) = \|A\|}}^{\hat{\kappa}(A,x)} \frac{\|x\|}{\|Ax\|} = \|A\| \frac{\|x\|}{\|Ax\|}$$

    - want an x ⫫ bound,
        - <u>Def</u>: the condition # of A, $\kappa(A) = \sup_{x \ne 0} \{\kappa(A,x)\} = \|A\| \sup_{x \ne 0} \left\{ \dfrac{\|x\|}{\|Ax\|} \right\}$

    - if $A \in \mathbb{C}^{m \times m}$ and is invertible then $\exists\, y$ s.t. $Ax = y \Leftrightarrow A^{-1}y = x$ for all x
      (if non-invertible use the psuedo-inverse, $A^{\dagger}$ over $y \in \text{range}(A)$)

        - then: $\kappa(A,x) = \|A\| \dfrac{\|x\|}{\|Ax\|} = \|A\| \dfrac{\|A^{-1}y\|}{\|y\|}$ so $\kappa(A) = \sup_{y \ne 0} \left\{ \|A\| \dfrac{\|A^{-1}y\|}{\|y\|} \right\} = \|A\| \sup_{y\ne 0} \left\{ \dfrac{\|A^{-1}y\|}{\|y\|} \right\} = \|A\|\,\|A^{-1}\|$

so, $\kappa(A,x) \leq \kappa(A)$ $\forall x$ where:

- the condition # of A is: $\kappa(A) = \|A\| \, \|A^{-1}\|$

  - **Ex**: using $\|\cdot\|_2$, $\|A\| = \sigma_{max}$, $\|A^{-1}\| = \max_j \{ 1/\sigma_1, 1/\sigma_2, \dots 1/\sigma_n \} = \frac{1}{\sigma_{min}}$

    so $\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}} \geq 1$ always lose accuracy...

- notice $\|A\| \, \|A^{-1}\|$ is symmetric under inversion of A

  so:

  $$\kappa(A) = \|A\| \, \|A^{-1}\| = \kappa(A^{-1})$$

- doing & undoing a linear transformation have the same worst case conditioning.

- **Thm**: given $A \in \mathbb{C}^{m \times m}$, invertible and $f(x) = Ax$, $f^{-1}(x) = A^{-1}x$ and a norm $\|\cdot\|$ on $\mathbb{C}^m$, then:

$$\left.\begin{aligned} \kappa(A,x) &= \|A\| \frac{\|x\|}{\|Ax\|} \leq \|A\| \, \|A^{-1}\| \\ \kappa(A^{-1},y) &= \|A^{-1}\| \frac{\|y\|}{\|A^{-1}y\|} \leq \|A^{-1}\| \, \|A\| \end{aligned}\right\} = \kappa(A)$$

thus: the worst case conditioning of f and f^{-1} over all inputs is $\kappa(A)$,

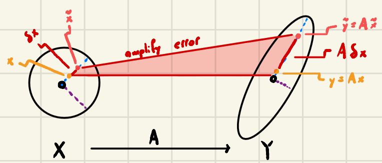
$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = O(\kappa(A,x)\,\varepsilon) \leq O(\kappa(A)\,\varepsilon)$$

$$\frac{\|\tilde{f}^{-1}(y) - f^{-1}(y)\|}{\|f^{-1}(y)\|} = O(\kappa(A^{-1},y)\,\varepsilon) \leq O(\kappa(A)\,\varepsilon)$$

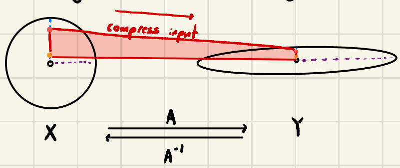
- if A^{-1} does not exist, use the psuedo-inverse, $\kappa(A) = \sigma_{max}/\sigma_{min \neq 0}$
- if use $\|\cdot\|_2$ achieve equality (worst case input) when x is v_{min}, y is u_{max}

- **Moral**: A is ill-conditioned if:

  1. $\|A\|$ is large $\Rightarrow$ amplifies some small input (small in, big out)



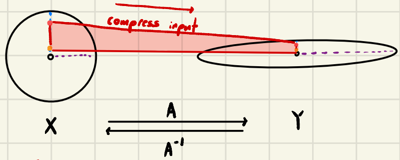
  2. $\|A^{-1}\|$ is large $\Rightarrow$ compresses a large input (big in, small out)



so, optimally conditioned, $\kappa(A) = 1$ if inputs don't change length ($\|Ax\| = \|x\|$ for all x... requires special type of A)

- **Question**: why does the reverse problem (solve $Ax = y$, apply A^{-1}) effect the conditioning of the forward problem (compute Ax)?

• **Question**: why does the reverse problem (solve $Ax=y$, apply A^{-1}) effect the conditioning of the forward problem (compute Ax)?



(• obvious for the inverse/reverse problem
 • if disparate inputs → similar outputs
 then similar outputs could come from disparate inputs... } leave for inverse problems unit)

• notice: compressing inputs requires combining large #'s to make small #'s (either divide by large #, multiply by small #, or subtract & nearly cancel) } numerically unstable provokes cancellation errors...

• so, arises from relative notion of error/number system

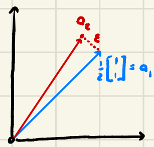
• recall:

$$\kappa(A)=\sup_{x\neq 0}\{\kappa(A,x)\}=\sup_{x\neq 0}\Big\{\underbrace{\hat{\kappa}(A,x)}_{\|A\|}\underbrace{\frac{\|x\|}{\|Ax\|}}_{\text{"compression factor of in}\to\text{out}}\Big\}=\|A\|\sup_{x\neq 0}\Big\{\frac{\|x\|}{\|Ax\|}\Big\}=\|A\|\,\|A^{-1}\|$$

from relative error

• Ex: $\|A^{-1}\|$ is large if col's of A are close to linearly dependent (parallel under combination) then large x can be mapped to small y by A...

let $A=\frac{1}{2}\begin{bmatrix}1 & 1-\varepsilon\\ 1 & 1+\varepsilon\end{bmatrix}$ for small $\varepsilon\neq 0$, $\varepsilon<1$.



• then A^{-1} exists. ill-conditioned for small ε... find large x s.t. Ax is very small (near 0, col's nearly cancel)

• pick: $x=\alpha\begin{bmatrix}1\\-1\end{bmatrix}$ then $\|x\|=\mathcal{O}(|\alpha|)$

$$Ax=\frac{\alpha}{2}\left(\begin{bmatrix}1\\1\end{bmatrix}-\begin{bmatrix}1-\varepsilon\\1+\varepsilon\end{bmatrix}\right)=\frac{\alpha}{2}\begin{bmatrix}\varepsilon\\-\varepsilon\end{bmatrix}\quad\text{so}\quad \|Ax\|=\mathcal{O}(|\alpha||\varepsilon|)$$

then: $\dfrac{\|x\|}{\|Ax\|}=\mathcal{O}(|\varepsilon^{-1}|)$ can make $\|A\|=\mathcal{O}(|\varepsilon|^{-1})$ arbitrarily large as $\varepsilon\to 0$

• ill-conditioned since the input $x\to Ax$ requires subtracting two large ($\alpha\gg\varepsilon$), similar #'s

• need $\frac{\alpha}{2}-\left(\frac{\alpha}{2}+\frac{\varepsilon\alpha}{2}\right)=\varepsilon\frac{\alpha}{2}$, ← requires cancellation.

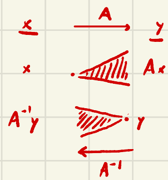
Thursday - 04/06/2023 - Orthogonality, Projection & Orthogonalization

- **Logistics:**
  - Reading, HW 2 & Project 1 posted
  - Today's lecture will come in 2 parts:
    - 12:30 - 1:20 today in class
    - last half recorded & posted to canvas

- **Goals:**
  - What transforms are optimally conditioned?
  - How do we optimize the conditioning of a basis?

  - leads to:
    - Orthonormal / Unitary matrices
    - Projection
    - Orthogonalization (Gram-Schmidt, mGS and QR)

- **Question:** if $\kappa(A) = \|A\|\,\|A^{-1}\|$, what A (square) are optimally conditioned?



  - $\|A\| = \sup\limits_{\|x\|=1} \{ \|Ax\| \}$ = "max amplification of length"
  - $\|A^{-1}\| = \sup\limits_{\|y\|=1} \{ \|A^{-1}y\| \} = \left[\inf\limits_{\|x\|=1} \{ \|Ax\| \} \right]^{-1}$ = "max compression of length"

so, $$\kappa(A) = \frac{\sup\limits_{\|x\|=1} \{\|Ax\|\}}{\inf\limits_{\|x\|=1} \{\|Ax\|\}} \geq 1$$

to acheive $\kappa(A) = 1$, we need $\sup\limits_{\|x\|=1} \{\|Ax\|\} = \inf\limits_{\|x\|=1} \{\|Ax\|\}$

$\therefore$ need $\|Ax\| = 1$ if $\|x\| = 1$ $\forall$ such x

that is $\|Ax\| / \|x\| = 1$ $\forall$ $x \neq 0$ $\Rightarrow$ $\|Ax\| = \|x\|$

so $\kappa(A) = 1$ if $T(x) = Ax$ preserves length... $\|Ax\| = \|x\|$ $\forall$ x

- using $\|\cdot\|_2$, $\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}} = 1$ if all $\sigma_j(A) = \sigma$ so, $A \cdot$ (unit ball) $\rightarrow$ ellipse w/ principle axes of length σ... the unit ball, $\frac{1}{\sigma}A$ must preserve length... $\underbrace{\|Ax\|_2 = \sigma\,\|x\|_2}$ $\forall$ x

- a very special class of A... <u>unitary matrices</u> (up to scaling)

Unitary / Orthonormal Matrices:

- **Def:** given an inner product $\langle \cdot , \cdot \rangle$ on a vector space V
  $u, v \in V$ are orthogonal w.r.t. $\langle \cdot , \cdot \rangle$ iff
  $$\langle u, v \rangle = 0$$
  - usually, $u, v \in \mathbb{C}^n$, $u^* v = 0$, interpret $u \underset{\text{"perpendicular"}}{\perp} v$

- **Def:** $A \in \mathbb{C}^{m \times n}$ $(m \geq n)$ is orthonormal (has $\perp$ norm col's) if:
  1. $\|a_i\| = 1 \; \forall i \quad (\|a_i\|^2 = \langle a_i, a_i \rangle)$ ← "normal"
  2. $a_i \perp a_j \; \forall i \neq j \in [1, n]$ ← mutually orthogonal.

  - **Def:** A is unitary if it is square & has $\perp$ norm col's

  - notation: often use Q for $\perp$ norm matrices

- Properties:
  1. $Q \in \mathbb{C}^{m \times n}$, $\perp$ norm col's iff $Q^* Q = I_{n \times n}$
     - why? $[Q^* Q]_{ij} = q_i^* q_j = \begin{cases} = 0 & \text{if } i \neq j \\ = 1 & \text{if } i = j \end{cases}$
  2. Q is unitary iff $Q^* = Q^{-1} \Rightarrow \left.\begin{array}{l} Q^* Q = I \\ Q Q^* = I. \end{array}\right\}$ exercise
  3. if Q is unitary, then $\kappa(Q) = 1$, $\kappa(Q^*) = 1$
     - why? ... consequence of ...
  4. if Q is unitary, then $T(x) = Qx$, $\langle T(x), T(y) \rangle = \langle x, y \rangle$
     $$\langle T(x), T(y) \rangle = \langle Qx, Qy \rangle = (Qx)^* Qy = x^* \underbrace{Q^* Q}_{I} y = x^* y = \langle x, y \rangle$$
     - preserve lengths and angles ... "rigid body" transformations
       rotations & reflections...

- the fact that, if Q has $\perp$ norm col's then $Q^*Q = I$ is very powerful...

  - **Ex:** suppose $Q \in \mathbb{C}^{m \times n}$, $Q = \{q_j\}_{j=1}^n$, where $\{q_j\}$ are a basis for a subspace $(S = \text{span}(\{q_j\}_{j=1}^n) = \text{range}(Q))$
    then $\forall\, x \in S \;\; \exists\, y \in \mathbb{C}^n$ s.t.

$$x = \sum_{j=1}^n y_j \, q_j = Q y \quad \text{for some } y \ldots$$

conversion $x \to y$, solve $Qy = x$

$$y = I y = Q^* Q y = Q^* x.$$

(entrywise: $y_j = [Q^* x]_j = q_j^* x = q_j^* \sum_{k=1}^n y_k q_k = \sum_{k=1}^n y_k \underbrace{\langle q_j, q_k \rangle}_{\substack{=0 \text{ if } j \neq k \\ =1 \text{ if } j = k}} = y_j$)

- expense: n inner products, vectors length m
  cost $\mathcal{O}(nm)$
  - relative to generic problem, solve $Ay = x$ for $A \in \mathbb{C}^{m \times n}$
    cubic in dim of A
- and, optimally condition

- motivates working w/ orthonormal matrices...

- begs the question, what is QQ^* given $Q \in \mathbb{C}^{m \times n}$, orthonormal?

$$Q = \underset{\leftarrow n \rightarrow}{m \updownarrow \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}}, \quad Q^* Q = \underset{n \times m}{\begin{bmatrix} \text{---} \\ \text{---} \end{bmatrix}} \underset{m \times n}{\begin{bmatrix} | & | & | \end{bmatrix}} = \underset{\leftarrow n \rightarrow}{\begin{bmatrix} \ddots & & \\ & I & \\ & & \ddots \end{bmatrix}} \updownarrow n, \quad QQ^* = \underset{m \times n}{\begin{bmatrix} | & | & | \end{bmatrix}} \underset{n \times m}{\begin{bmatrix} \text{---} \\ \text{---} \end{bmatrix}} = \begin{bmatrix} & ? & \end{bmatrix}$$

QQ^* is the orthogonal projector onto $\text{range}(Q) = \text{span}\{q_j\}_{j=1}^n$.

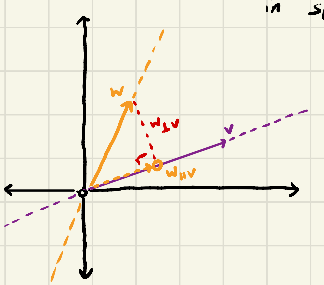
- why? well, easiest case, $Q \in \mathbb{C}^{m \times 1}$, $Q = \begin{bmatrix} q_1 \end{bmatrix}$

$$QQ^* = q_1 \; q_1^* = m \updownarrow \underset{\leftarrow m \rightarrow}{\begin{bmatrix} P_{\parallel Q} \end{bmatrix}}, \quad x_{\parallel q_1} = \frac{(q_1^* x)}{\|q_1\|^2} q_1 = (q_1 q_1^*) x = P_{\parallel Q}\, x$$

Projection, a review:

Projection:

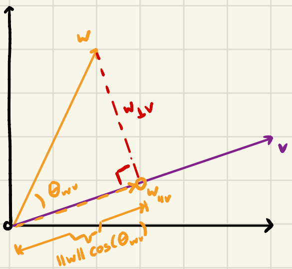
- **Projection onto Lines**: say we are given a line $\parallel v$, and a point in space specified w



projection of w onto v = a point on the line $\parallel v$ "where w casts it's shadow"

if $w_{\parallel v}$ is the projection of w onto v then the triangle formed by w, $w_{\parallel v}$, origin is a right triangle

- how to compute $w_{\parallel v}$?



- **notice**: $w = w_{\parallel v} + w_{\perp v}$, $w_{\parallel v} \perp w_{\perp v}$

$w_{\perp v} = w - w_{\parallel v}$

well, $w_{\parallel v}$ is $\parallel$ to v so $w = \alpha \hat{v} = \alpha \frac{v}{\|v\|}$ for some scalar α.

trig:

$$\|w_{\parallel v}\| = \|w\| \cos(\theta_{wv})$$

since $w_{\parallel v}$, $w_{\perp v}$ and v form a right triangle w/ hypotenuse length $\|w\|$ adjacent side length $\|w_{\parallel v}\|$ and angle θ_{wv}

$$\|w_{\parallel v}\| = \|\alpha \hat{v}\| = |\alpha| \, \underbrace{\|\hat{v}\|}_{=1,\text{ unit vector}} = |\alpha|$$

$$\therefore \quad |\alpha| = \|w\| \cos(\theta_{wv}) = \|w\| \frac{v^T w}{\|v\| \|w\|} = \frac{v^T w}{\|v\|}$$

so $$w_{\parallel v} = \alpha \hat{v} = \left(\frac{v^T w}{\|v\|}\right) \frac{v}{\|v\|} = \left(\frac{v^T w}{\|v\|^2}\right) v = \underbrace{\left(\frac{v^T w}{v^T v}\right)}_{\text{a scalar}} v$$

$$= v \left(\frac{v^T w}{v^T v}\right) = \left(\frac{v v^T}{v^T v}\right) w$$

$$= \left(\frac{1}{\|v\|^2} v v^T\right) w = \underbrace{(\hat{v} \hat{v}^T)}_{} w$$

$P_{\parallel v}$ = matrix that projects onto v

- **Projector onto v**: $P_{\parallel v} = (\hat{v} \hat{v}^T) = \frac{1}{\|v\|^2} (v v^T)$
- **Projector $\perp$ to v**: $P_{\perp v} = I - P_{\parallel v}$ since $w_{\perp v} = w - w_{\parallel v} = I w - P_{\parallel v} w = (I - P_{\parallel v}) w$

• Constructing ⊥ Projectors:

• **onto a line v:** $P_{\parallel v} = \hat{v}\hat{v}^T = \frac{1}{\|v\|^2}(vv^T)$ ⟵ let's check:

⊥ projector onto range(v)

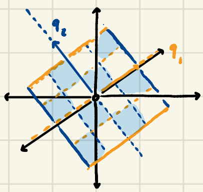
• $(\hat{v}\hat{v}^T)^2 = \hat{v}\hat{v}^T\hat{v}\hat{v}^T = \hat{v}(\underbrace{\hat{v}^T\hat{v}}_{\|\hat{v}\|^2=1})\hat{v}^T$

$= \hat{v}\cdot 1\,\hat{v}^T = \hat{v}\hat{v}^T$

• $(\hat{v}\hat{v}^T)^T = (\hat{v}^T)^T\hat{v}^T = \hat{v}\hat{v}^T$

• **onto a subspace $\mathcal{V}$:** let $Q_{\mathcal{V}} = [q_1, q_2, \dots q_d]$ be an ⊥ matrix whose columns form a basis for $\mathcal{V} \in \mathbb{R}^m$

then $\text{range}(Q_{\mathcal{V}}) = \mathcal{V}$.

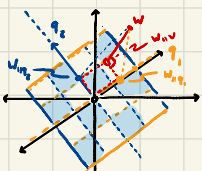


range $(Q) = \mathcal{V}$

we can project onto each basis vector q_j

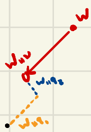
via $P_{\parallel q_j} = \frac{1}{\|q_j\|^2}(q_j q_j^T) = q_j q_j^T$

so $w_{\parallel q_j} = (q_j q_j^T)w$



range $(Q) = \mathcal{V}$

$w_{\parallel \mathcal{V}}$ is in $\mathcal{V} = \text{span}(\{q_1, \dots q_d\})$ so $w_{\parallel \mathcal{V}}$ is a linear combination of the basis vectors...



$$w_{\parallel \mathcal{V}} = y_1 q_1 + y_2 q_2 + \dots y_d q_d = Q_{\mathcal{V}} y$$

$$= \underbrace{w_{\parallel q_1} + w_{\parallel q_2} + \dots w_{\parallel q_d}}_{\text{components of } w \parallel \text{ to each basis vector}}$$

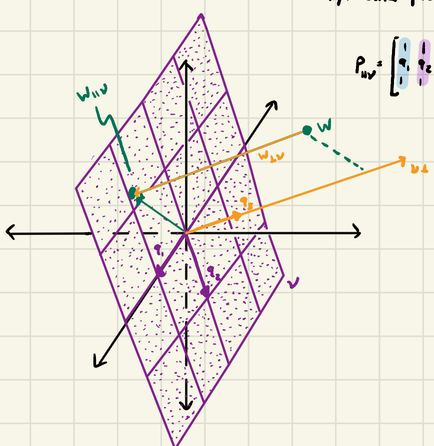
then: $w_{\parallel \mathcal{V}} = (q_1 q_1^T)w + (q_2 q_2^T)w + \dots (q_d q_d^T)w$

$= (q_1 q_1^T + q_2 q_2^T + \dots q_d q_d^T)w$

so: $P_{\parallel \mathcal{V}} = \sum_{j=1}^{d} q_j q_j^T = q_1 q_1^T + q_2 q_2^T + \dots q_d q_d^T$

or, more concisely: $P_{\parallel \mathcal{V}} = Q_{\mathcal{V}} Q_{\mathcal{V}}^T$

• why? outer-product convention



$$P_{\parallel \mathcal{V}} = \begin{bmatrix} q_1 & q_2 & \cdots & q_d \end{bmatrix}\begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_d^T \end{bmatrix} = q_1 q_1^T + q_2 q_2^T + \dots q_d q_d^T$$

$$= \sum_{j=1}^{d} q_j q_j^T$$

so: $P_{\parallel \mathcal{V}} w = \sum_{j=1}^{d} (q_j q_j^T w) = \sum_{j=1}^{d} q_j (q_j^T w) = \sum_{j=1}^{d} \overbrace{(q_j^T w)}^{\text{"}y_j\text{"}} q_j$

compare to:

let $Q = [Q_{\mathcal{V}} \,|\, Q_{\perp \mathcal{V}}]$ be ⊥ and $n \times n$, then $\text{range}(Q) = \mathbb{R}^m$

$w = Qy$ for some y. Q is ⊥ so $y = Q^T w$

$y_j = (q_j^T w)$ ∴ $w = \sum_{j=1}^{m} (q_j^T w) q_j = \underbrace{\left[\sum_{j=1}^{d} (q_j^T w) q_j\right]}_{w_{\parallel \mathcal{V}}} + \underbrace{\left[\sum_{j=d+1} (q_j^T w) q_j\right]}_{w_{\perp \mathcal{V}}}$

• <u>why does this work?</u>

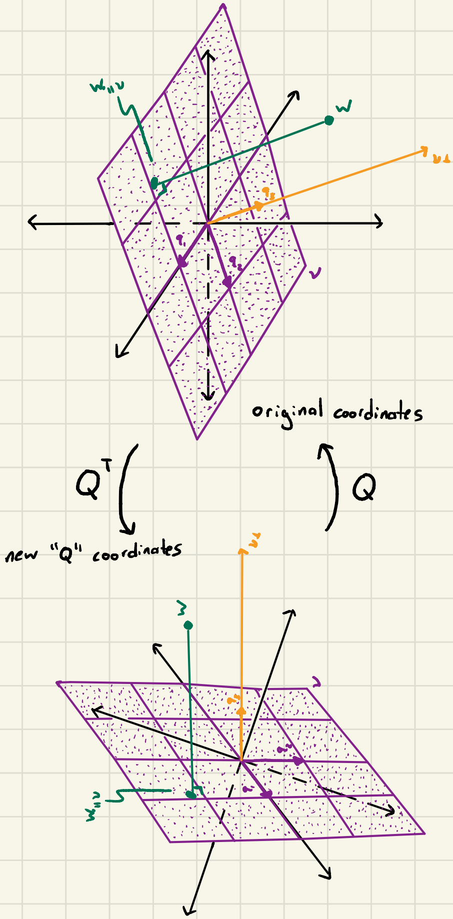
• Suppose we have a subspace $\mathcal{V}$

how do we build $P_{\mathcal{V}}$ the orthogonal projector onto $\mathcal{V}$

• <u>idea</u>: suppose $\mathcal{V} \in \mathbb{R}^m$, then let $Q_{\mathcal{V}}$ be an $\perp$ basis for $\mathcal{V}$, let $\mathcal{V}^\perp$ be the subspace $\perp$ $\mathcal{V}$, let $Q_{\mathcal{V}\perp}$ be $\perp$ basis for $\mathcal{V}^\perp$

then build Q $\perp$ basis $\mathbb{R}^m$

$$Q = \left[\, Q_{\mathcal{V}} \;\vdots\; Q_{\mathcal{V}\perp} \,\right]$$



• now, Q is $m \times m$ & orthonormal so we can multiply by Q^T to change coordinates

$$w \xmapsto{\text{change coord.}} Q^T w = \begin{bmatrix} Q_{\mathcal{V}}^T \\ \hline Q_{\mathcal{V}\perp}^T \end{bmatrix} w = \begin{bmatrix} Q_{\mathcal{V}}^T w \\ \hline Q_{\mathcal{V}\perp}^T w \end{bmatrix}$$

coordinates associated w/ $\mathcal{V}$

w in the "Q" coordinate system

coordinates associated w/ $\mathcal{V}^\perp$

in this coordinate system $w_{\parallel\mathcal{V}} \longmapsto$ the first $\dim(\mathcal{V})$ components of $Q^T w$

here $\perp$ projection onto $\mathcal{V}$ is easy

$$\begin{bmatrix} Q_{\mathcal{V}}^T w \\ \hline Q_{\mathcal{V}\perp}^T w \end{bmatrix} \xrightarrow{\text{project}} \begin{bmatrix} Q_{\mathcal{V}}^T w \\ \hline 0 \; w \end{bmatrix}$$

just set $\perp$ component to zero.

but now, moving back to my original coordinates is also easy since I just multiply by Q

$$w_{\parallel\mathcal{V}} = Q \begin{bmatrix} Q_{\mathcal{V}}^T w \\ \hline 0 \end{bmatrix} = \left[\, Q_{\mathcal{V}} \;\vdots\; Q_{\mathcal{V}\perp} \,\right] \begin{bmatrix} Q_{\mathcal{V}}^T w \\ \hline 0 \end{bmatrix} = Q_{\mathcal{V}} Q_{\mathcal{V}}^T w + \overbrace{Q_{\mathcal{V}\perp} \cdot 0}^{=0}$$

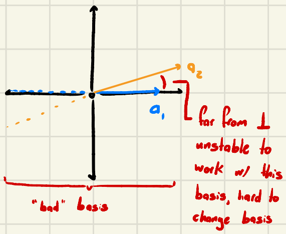
$$= Q_{\mathcal{V}} Q_{\mathcal{V}}^T w$$

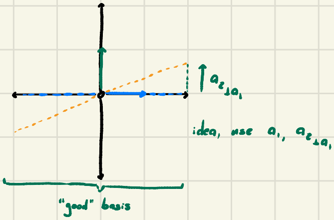
basis for $\mathbb{R}^m$ that works nicely w/ $\mathcal{V}$

$$w_{\parallel\mathcal{V}} = (Q_{\mathcal{V}} Q_{\mathcal{V}}^T)\, w$$

so: $P_{\parallel\mathcal{V}} = Q_{\mathcal{V}} Q_{\mathcal{V}}^T$

- **Problem:** given a basis $\{a_j\}_{j=1}^{n}$ $\left(= \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} = A\right)$, can we combine the basis vectors (col's of A) to produce an $\perp$ norm. basis that spans $S = \mathrm{span}(\{a_j\}_{j=1}^{n}) = \mathrm{range}(A)$?





- **Solution:** set:

$$\left.\begin{aligned} q_1 &\parallel a_1 \\ q_2 &\parallel a_{2\perp a_1} = a_{2\perp q_1} \\ q_3 &\parallel a_{3\perp a_1,a_2} = a_{3\perp q_1,q_2} \\ &\;\vdots \\ q_j &\parallel a_{j\perp a_1,\dots,a_j} = a_{j\perp q_1,\dots,q_j} \end{aligned}\right\}$$

write: $A^{(j)} = \begin{bmatrix} a_1 & a_2 & \cdots & a_j \end{bmatrix}$, $Q^{(j)} = \begin{bmatrix} q_1 & q_2 & \cdots & q_j \end{bmatrix}$

where $Q^{(j)}$ $\perp$ norm and $\mathrm{range}(Q^{(j)}) = \mathrm{range}(A^{(j)})$

compute: $$a_{j\perp Q^{(j-1)}} = P_{\perp Q^{(j-1)}} a_j = (I - P_{\parallel Q^{(j-1)}}) a_j = (I - Q^{(j-1)} Q^{(j-1)*}) a_j$$

$$= a_j - \left[\sum_{k=1}^{j-1} q_k q_k^*\right] a_j = a_j - \sum_{k=1}^{j-1} (\underbrace{q_k^* a_j}_{``r_{kj}"}) q_k = a_j - \sum_{k=1}^{j-1} r_{kj} q_k$$

- **Algorithm:** Gram-Schmidt

- given $A \in \mathbb{C}^{m\times n}$
- initialize $Q^{(0)} = [\;]^{m\times n}$, $R^{(0)} = [\;]^{n\times n}$ ← empty, will fill as we go

for $j = 1:n$

(i) propose a candidate direction: $v = a_j$

(ii) orthogonalize: for $k = 1 : j-1$

(a) $r_{kj} = q_k^* a_j$

(b) $v = v - r_{kj} q_k$

(iii) normalize:

(a) $r_{jj} = \|v\|$

(b) $q_j = \dfrac{v}{r_{jj}}$

(iv) store:

$Q^{(j)} = \begin{bmatrix} Q^{(j-1)} & q_j \end{bmatrix}$, $R^{(j)} = \left[\begin{array}{c|c} R^{(j-1)} & \begin{matrix} r_{1j} \\ r_{2j} \\ \vdots \end{matrix} \\ \hline 0 & r_{jj} \end{array}\right]$

- **Algorithm: Gram-Schmidt**

  - given $A \in \mathbb{C}^{m \times n}$
  - initialize $Q^{(0)} = [\ ]^{m \times n}$, $R^{(0)} = [\ ]^{n \times n}$ ← empty, will fill as we go

alternate algorithm: $r_{kj} = q_k^* v$

for $j = 1 : n$

(i) propose a candidate direction: $v = a_j$

(ii) orthogonalize: for $k = 1 : j-1$

(a) $r_{kj} = q_k^* a_j$ ← equals $q_k^* v$

(b) $v = v - r_{kj} q_k$

computes: $a_{\perp Q^{(j-1)}} = v_{\perp Q^{(j-1)}} = P_{\perp Q^{(j-1)}} v$

$= (I - P_{\parallel Q^{(j-1)}}) v = (I - Q^{(j-1)} Q^{(j-1)*}) v$

$= v - \sum_{k=1}^{j-1} q_k \underbrace{(q_k^* v)}_{r_{kj}} = v - \sum_{k=1}^{j-1} r_{kj} q_k.$

(iii) normalize:

(a) $r_{jj} = \|v\|$ ← equals $\|a_{k_{\perp Q^{(k-1)}}}\|$

(b) $q_j = \dfrac{v}{r_{jj}}$

implies: $a_{\perp Q^{(j-1)}} = a - a_{\parallel Q^{(j-1)}} = a - \sum_{k=1}^{j-1} r_{kj} q_k$

so: $a_j = a_{\perp Q^{(j-1)}} + \sum_{k=1}^{j-1} r_{kj} q_k$

so: $a_{j_{\perp Q^{(j-1)}}} = v = r_{jj} q_j$

$a_j = r_{jj} q_j + \sum_{k=1}^{j-1} r_{kj} q_k$

$a_j = \sum_{k=1}^{j} q_k r_{kj}$

$$\begin{bmatrix} a_j \end{bmatrix} = \underbrace{\begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}}_{Q} \begin{bmatrix} r_{1j} \\ \vdots \\ r_{jj} \\ 0 \end{bmatrix}$$

j^{th} col of R

(iv) store:

$$Q^{(j)} = [Q^{(j-1)}, q_j], \quad R^{(j)} = \left[\begin{array}{c|c} R^{(j-1)} & \begin{matrix} r_{1j} \\ r_{2j} \end{matrix} \\ \hline 0 & r_{jj} \end{array}\right]$$

- implies a decomposition of A...

so: $A = QR = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \begin{bmatrix} R \end{bmatrix}$ (m rows; R is $n \times n$)

where: Q orthonormal $m \times n$

R upper triangular, $n \times n$

- **QR decomposition**: given $A \in \mathbb{C}^{m \times n}$, linearly independent col's $(\text{rank}(A) = n \le m)$ then $\exists$ Q and R s.t.

$$A = QR \quad \text{where } Q \in \mathbb{C}^{m \times n} \text{ has orthonormal col's}$$
$$\text{and } R \in \mathbb{C}^{n \times n} \text{ is upper triangular.}$$

"undo the orthogonalization"

- **Very powerful idea**: triangular orthogonalization only required inner products \$ lin. comb. converts a set of n linearly $\perp\!\!\!\perp$ vectors "A" to n $\perp$, normalized vectors "Q" and, by storing the inner products involved in R, we can express all vectors in A, say, a_j as a linear comb. of the preceding q's $(a_j = \sum_{k \le j} r_{kj} q_k)$

  - Works in any vector space equipped w/ an inner product!

- **Ex:** Gram-Schmidt for polynomials

$P_\Omega^{(m)} = \{$ all polynomials of degree $\leq m$ on $\Omega = [a, b] \}$

given $p, q \in P_\Omega^{(m)}$, $\langle p, q \rangle = \int_{x \in \Omega} \bar{p}(x)\, q(x)\, dx$

$\|p\|^2 = \langle p, p \rangle$

then can run GS to convert a set of $n \leq m$, lin. ⫫ polynomials

$$A = \{a_1(x), a_2(x), a_3(x), \dots a_n(x)\} \xrightarrow[GS]{} Q = \{q_1(x), q_2(x), \dots q_n(x)\}$$

s.t. $q_i(x) \perp q_j(x) \;\; \forall\, i \neq j$

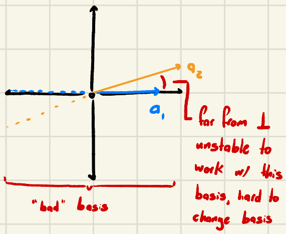
$\|q_i\| = 1 \;\; \forall\, i$

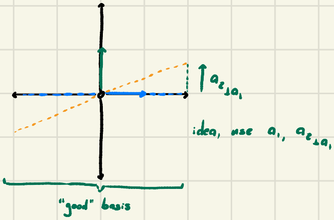
$\Bigg\}$ conditioning $\kappa(Q) = 1 \dots$

- Ex: $\underbrace{\{1, \; x, \; x^2, \; \dots \; x^n\}}_{\text{typically very ill-conditioned}}$

and $a_j(x) = \sum_{k \leq j} r_{kj}\, q_k(x)$ where r_{kj} follow from the inner-prod. computed via Gram-Schmidt.

- **Problem**: given a basis $\{a_j\}_{j=1}^{n}$ $\left(= \begin{bmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{bmatrix} = A\right)$, can we combine the basis vectors (col's of A) to produce an $\perp$ norm. basis that spans $S = \text{span}(\{a_j\}_{j=1}^{n}) = \text{range}(A)$?





- **Solution**: set:

$$\left.\begin{array}{lll} q_1 & \parallel & a_1 \\ q_2 & \parallel & a_{2_{\perp a_1}} = a_{2_{\perp q_1}} \\ q_3 & \parallel & a_{3_{\perp a_1, a_2}} = a_{3_{\perp q_1, q_2}} \\ & \vdots & \\ q_j & \parallel & a_{j_{\perp a_1, \ldots a_j}} = a_{j_{\perp q_1, \ldots q_j}} \end{array}\right\}$$

write: $A^{(j)} = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_j \\ | & | & & | \end{bmatrix}$, $Q^{(j)} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_j \\ | & | & & | \end{bmatrix}$

where $Q^{(j)}$ $\perp$ norm and $\text{range}(Q^{(j)}) = \text{range}(A^{(j)})$

compute: $a_{j_{\perp Q^{(j-1)}}} = P_{\perp Q^{(j-1)}} a_j = (I - P_{\parallel Q^{(j-1)}}) a_j = (I - Q^{(j-1)} Q^{(j-1)*}) a_j$

$$= a_j - \left[\sum_{k=1}^{j-1} q_k q_k^*\right] a_j = a_j - \sum_{k=1}^{j-1} \underbrace{(q_k^* a_j)}_{\text{"}r_{kj}\text{"}} q_k = a_j - \sum_{k=1}^{j-1} r_{kj} q_k$$

- **Algorithm**: Gram-Schmidt

- given $A \in \mathbb{C}^{m \times n}$
- initialize $Q^{(0)} = [\;]^{m \times n}$, $R^{(0)} = [\;]^{n \times n}$ ← empty, will fill as we go

for $j = 1:n$

(i) propose a candidate direction: $v = a_j$

(ii) orthogonalize: for $k = 1 : j-1$

(a) $r_{kj} = q_k^* a_j$

(b) $v = v - r_{kj} q_k$

(iii) normalize:

(a) $r_{jj} = \|v\|$

(b) $q_j = \dfrac{v}{r_{jj}}$

(iv) store:

$Q^{(j)} = [Q^{(j-1)}, q_j]$, $R^{(j)} = \left[\begin{array}{c|c} R^{(j-1)} & \begin{array}{c} r_{1j} \\ r_{2j} \\ \vdots \end{array} \\ \hline 0 & r_{jj} \end{array}\right]$

- **Algorithm: Gram-Schmidt**

  - given $A \in \mathbb{C}^{m \times n}$
  - initialize $Q^{(0)} = [\ ]^{m \times n}$, $R^{(0)} = [\ ]^{n \times n}$ ← empty, will fill as we go

for $j = 1 : n$

(i) propose a candidate direction: $v = a_j$

(ii) orthogonalize: for $k = 1 : j-1$

(a) $r_{kj} = q_k^* a_j$ ⟵ equals $q_k^* v$

alternate algorithm: $r_{kj} = q_k^* v$

(b) $v = v - r_{kj} q_k$

computes: $a_{\perp Q^{(j-1)}} = v_{\perp Q^{(j-1)}} = P_{\perp Q^{(j-1)}} v$

$= \left(I - P_{\parallel Q^{(j-1)}}\right) v = \left(I - Q^{(j-1)} Q^{(j-1)*}\right) v$

$= v - \sum_{k=1}^{j-1} q_k \underbrace{(q_k^* v)}_{r_{kj}} = v - \sum_{k=1}^{j-1} r_{kj} q_k .$

(iii) normalize:

(a) $r_{jj} = \|v\|$ ⟵ equals $\| a_{k_{\perp Q^{(k-1)}}} \|$

implies: $a_{\perp Q^{(j-1)}} = a - a_{\parallel Q^{(j-1)}} = a - \sum_{k=1}^{j-1} r_{kj} q_k$

(b) $q_j = \dfrac{v}{r_{jj}}$

so: $a_j = a_{\perp Q^{(j-1)}} + \sum_{k=1}^{j-1} r_{kj} q_k$

so: $a_{j_{\perp Q^{(j-1)}}} = v = r_{jj} q_j$

$a_j = r_{jj} q_j + \sum_{k=1}^{j-1} r_{kj} q_k$

$a_j = \sum_{k=1}^{j} q_k r_{kj}$

(iv) store:

$$Q^{(j)} = \left[Q^{(j-1)},\ q_j \right], \quad R^{(j)} = \begin{bmatrix} R^{(j-1)} & \begin{matrix} r_{1j} \\ r_{2j} \\ \vdots \end{matrix} \\ 0 & r_{jj} \end{bmatrix}$$

- implies a decomposition of A...

$$\big[a_j \big] = \underbrace{\begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}}_{Q} \begin{bmatrix} r_{1j} \\ \vdots \\ r_{jj} \\ 0 \end{bmatrix}$$

j^{th} col of R

so: $A = Q R = \underset{m}{\updownarrow}\begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \underset{\leftarrow n \rightarrow}{\begin{bmatrix} R \end{bmatrix}} \updownarrow n$

where: Q orthonormal $m \times n$

R upper triangular, $n \times n$

- **QR decomposition**: given $A \in \mathbb{C}^{m \times n}$, linearly independent col's $(\operatorname{rank}(A) = n \leq m)$
  then $\exists$ Q and R s.t.

$$A = QR \quad \text{where} \quad Q \in \mathbb{C}^{m \times n} \text{ has orthonormal col's}$$
$$\text{and} \quad R \in \mathbb{C}^{n \times n} \text{ is upper triangular.}$$

"undo the orthogonalization"

- **Very powerful idea**: triangular orthogonalization only required inner products \$ lin. comb.
  converts a set of n linearly ⫫ vectors "A" to n ⊥, normalized vectors "Q"
  and, by storing the inner products involved in R, we can express all vectors in
  A, say, a_j as a linear comb. of the preceding q's $\left(a_j = \sum_{k \leq j} r_{kj} q_k \right)$
  - Works in any vector space equipped w/ an inner product!

- **Ex:** Gram-Schmidt for polynomials

$P_\Omega^{(m)} = \{\text{all polynomials of degree} \le m \text{ on } \Omega = [a,b]\}$

given $p, q \in P_\Omega^{(m)}$, $\langle p, q\rangle = \int_{x\in\Omega} \bar{p}(x)\, q(x)\, dx$

$\|p\|^2 = \langle p, p\rangle$

then can run GS to convert a set of $n \le m$, lin. ⫫ polynomials

$$A = \{a_1(x), a_2(x), a_3(x), \dots a_n(x)\} \xrightarrow[GS]{} Q = \{q_1(x), q_2(x), \dots q_n(x)\}$$

- Ex: $\underbrace{\{1, x, x^2, \dots x^n\}}_{\text{typically very ill-conditioned}}$

s.t. $q_i(x) \perp q_j(x) \ \forall \ i \ne j$

$\|q_i\| = 1 \ \forall \ i$

conditioning $\kappa(Q) = 1$...

and $a_j(x) = \sum_{k\le j} r_{kj}\, q_k(x)$ where r_{kj} follow from the inner-prod. computed via Gram-Schmidt.

- **modified GS:** same as GS, but, compute $r_{kj} = q_k^* a_j$ differently...

- **Algorithm:** **Mod. Gram-Schmidt**

  - given $A \in \mathbb{C}^{m\times n}$
  - initialize $Q^{(0)} = [\ ]^{m\times n}$, $R^{(0)} = [\ ]^{n\times n}$

for $j = 1 : n$

(i) propose a candidate direction: $v = a_j$

(ii) orthogonalize: for $k = 1 : j-1$

(a) $r_{kj} = q_k^* v$ ⟵ compare to GS, $r_{kj} = q_k^* a_j$

(b) $v = v - r_{kj} q_k$

these are the same (theoretically) since, at stage k

$$v = a_j - \sum_{h=1}^{k-1} r_{hj} q_h \quad \text{so: } q_k^* v = q_k^* a_j - \sum_{h=1}^{k-1} r_{hj} \overbrace{q_k^* q_h}^{q_k \perp q_h = 0} = q_k^* a_j$$

(iii) normalize:

(a) $r_{jj} = \|v\|$

(b) $q_j = \dfrac{v}{r_{jj}}$

(iv) store:

$$Q^{(j)} = [Q^{(j-1)}, q_j], \quad R^{(j)} = \left[\begin{array}{c|c} R^{(j-1)} & \begin{matrix} r_{1j} \\ r_{2j} \\ \vdots \end{matrix} \\ \hline 0 & r_{jj} \end{array}\right]$$

allows us to reorder the loops, every time we compute a q_k remove it's component from all col's of A left to orthogonalize (all $a_{j>k}$)

- **Algorithm:** mod. Gram-Schmidt

  - given $A \in \mathbb{C}^{m \times n}$
  - initialize $Q^{(0)} = [\ ]^{m \times n}$, $R^{(0)} = [\ ]^{n \times n}$

1. set $V = A$
2. for $j = 1$ to n

   (i) $r_{jj} = \| v_j \|$

   (ii) $q_j = \frac{1}{r_{jj}} v_j$

   normalize

   (iii) for $k = j+1$ to n

   (a) $r_{jk} = q_j^* v_k$

   (b) $v_k = v_k - r_{jk} q_j$

   orthogonalize

   (iv) store: $Q^{(j)} = \left[Q^{(j-1)} \mid q_j \right]$, $R^{(j)} = \begin{bmatrix} & R^{(j-1)} & & & \\ \hline 0 \cdots 0 & r_{jj} & \cdots & r_{jn} \end{bmatrix}$ ← build R one row at a time

Week 4 - Orthogonalization & Fast Transforms. (inverse problems, linear systems, least squares)

Logistics:

- Reading & Project 1 posted
- HW 2 due on Thursday
- HW 3 to post Wednesday

Goals: 2 part lecture:

(i) Householder...

(ii) intro to fast transforms...

- **Householder:** like GS, given a $A \in \mathbb{C}^{m \times n}$ ($n \leq m$), convert $A \to Q, R$ s.t. $A = QR$

where: Q is orthonormal $\leftarrow$ $\underbrace{m \times n}$, $\underbrace{m \times m}$

R is upper triangular $\leftarrow$ $\underbrace{n \times n}_{\hat{Q}, \hat{R}}$, $\underbrace{m \times n}_{Q, R}$

  - **GS:** $A \xrightarrow{\text{col op}} \hat{Q}$, triangular orthogonalization

    reduction via a triangular matrix (recorded by R)

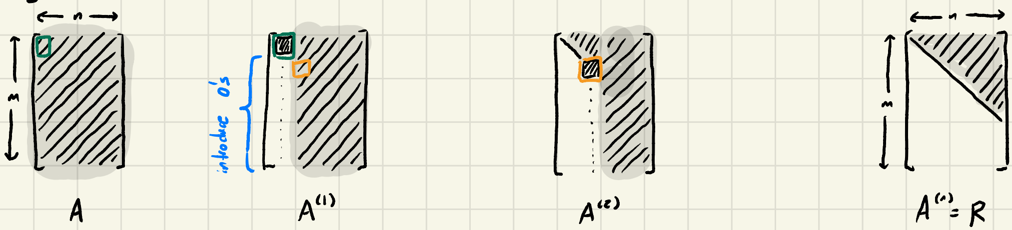
  - **Householder:** $A \xrightarrow{\text{row op}} R$

    reduction via an orthonormal, unitary matrix (Q^*, record Q)

  - **Take-aways:**

    1. derivation process
    2. block matrices & block products
    3. implicit representation of operations
    4. mathematically equivalent methods can be numerically distinct

• Deriving Householder: reduce $A \to R$ via unitary operations (row operations)



$$A^{(0)} \xmapsto{Q^{(1)}} Q^{(1)}A^{(0)} = A^{(1)} \xmapsto{Q^{(2)}} Q^{(2)}A^{(1)} = A^{(2)} \longmapsto \cdots \longrightarrow Q^{(n)}A^{(n-1)} = A^{(n)} = R$$

recursion: $A^{(j)} = Q^{(j)}A^{(j-1)}$, $A^{(0)} = A$, $A^{(n)} = R$

$$A^{(j)} = Q^{(j)}Q^{(j-1)}Q^{(j-2)} \cdots Q^{(2)}Q^{(1)}A^{(0)}$$

$$A^{(n)} = \underbrace{\left[Q^{(n)}Q^{(n-1)} \cdots Q^{(2)}Q^{(1)}\right]}_{Q^*} A = R,$$

$Q^* A = R$

$Q Q^* A = QR$

$A = QR$

$Q = Q^{(1)*}Q^{(2)*} \cdots Q^{(n-1)*}Q^{(n)*}$.

Q^* explicitly maps $A \mapsto R$

Q ... $R \mapsto A$

• goal: build the $\perp$ projector $P_{\parallel A} = \hat{Q}\hat{Q}^*$

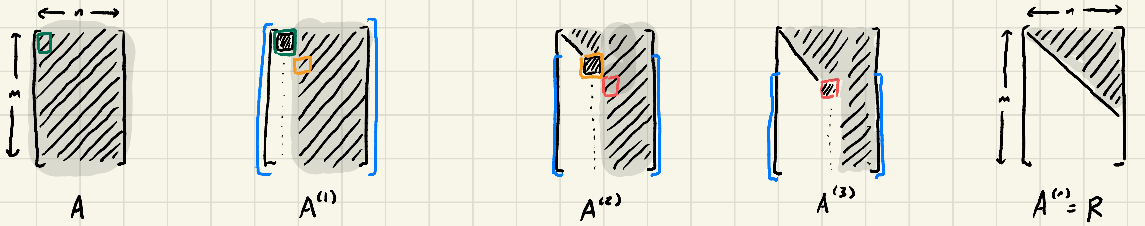
applying $x_{\parallel A} = P_{\parallel A}\, x = \hat{Q}(\hat{Q}^* x)$

• Ex: store the minimal information needed to apply the transform $T^{(j)}(x) = Q^{(j)}x$

$$\begin{aligned}\hat{Q}^* x &= Q^{(n)}Q^{(n-1)} \cdots Q^{(2)}Q^{(1)}x \\ &= T^{(n)}(T^{(n-1)}(\cdots T^{(1)}(x))) \\ &= T^{(n)} \circ T^{(n-1)} \cdots \circ T^{(2)} \circ T^{(1)}(x).\end{aligned}$$

perform recursively

$x^{(0)} = x$, $x^{(j)} = T^{(j)}(x^{(j-1)})$ ← multiplication by Q^* implicitly.



$$A^{(0)} \xmapsto{\;Q^{(1)}\;} Q^{(1)}A^{(0)} = A^{(1)} \xmapsto{\;Q^{(2)}\;} Q^{(2)}A^{(1)} = A^{(2)} \longmapsto \cdots \longmapsto Q^{(n)}A^{(n-1)} = A^{(n)} = R$$

acts on all rows of A

only change rows $2 \dots m$

$Q^{(3)}$ will only change rows $3 \dots m$

design: $Q^{(j)}$ only change rows $j \dots m$ of input.

method: block construction of $Q^{(j)}$:

$$Q^{(j)} = \underbrace{\begin{bmatrix} I_{(j-1)} & 0 \\ 0 & F^{(j)} \end{bmatrix}}_{m \times m}, \quad F^{(j)} : (m-(j-1)) \times (m-(j-1))$$

} leave 1st $j-1$ rows of $A^{(j-1)}$ alone

- block multiplication by matrices:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad M' = \begin{bmatrix} A' & B' \\ C' & D' \end{bmatrix}$$

$$MM' = \begin{bmatrix} AA' + BC' & AB' + BD' \\ CA' + DC' & CB' + DD' \end{bmatrix}$$

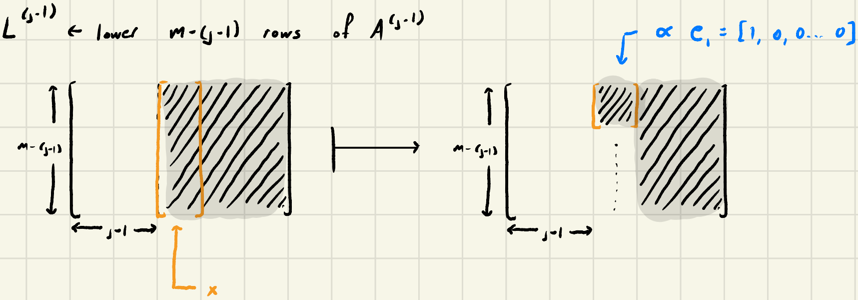
- check: $Q^{(j)} = \left[\begin{array}{c|c} I_{j-1 \times j-1} & 0 \\ \hline 0 & F^{(j)} \end{array}\right]$, $\quad A^{(j-1)} = \begin{array}{c} \left[\begin{array}{c} U^{(j-1)} \\ \hline L^{(j-1)} \end{array}\right] \end{array} \begin{array}{l} \updownarrow j-1 \\ \updownarrow m-(j-1) \end{array}$

$$Q^{(j)}A^{(j-1)} = \begin{bmatrix} I & 0 \\ 0 & F^{(j)} \end{bmatrix}\begin{bmatrix} U^{(j-1)} \\ L^{(j-1)} \end{bmatrix} = \begin{bmatrix} I_{j-1 \times j-1}U^{(j-1)} + 0 \\ 0 + F^{(j)}L^{(j-1)} \end{bmatrix} = \left[\begin{array}{c} U^{(j-1)} \\ \hline F^{(j)}L^{(j-1)} \end{array}\right] \begin{array}{l} \updownarrow j-1 \\ \updownarrow m-(j-1) \end{array}$$

} left 1st $j-1$ rows alone!

- build $F^{(j)}$ (reflectors)

at stage $F^{(j)}$, $L^{(j-1)} \leftarrow$ lower $m-(j-1)$ rows of $A^{(j-1)}$



- def: at stage j, x be the last $m-(j-1)$ entries of j^{th} col. of $A^{(j-1)}$

$$x = A^{(j-1)}_{(j:m),j}$$

take rows j to m of j^{th} col. of $A^{(j-1)}$

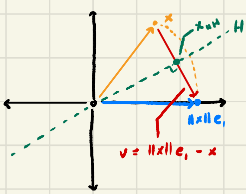
- general problem: find a unitary matrix $F^{(j)} \in \mathbb{C}^{m-(j-1) \times m-(j-1)}$

s.t.

$$F^{(j)} x = z \|x\| \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = z \|x\| e_1$$

require $z \in \mathbb{C}$, $|z| = 1$

- for simplicity... pick $z = 1$ (for now)



- unitary transformations: rotations & reflections

- def: let $v = \|x\| e_1 - x$

  let $H = \{ y \mid y \perp v \}$

reflection across H sends $x \to \|x\| e_1$

going "halfway", $x_{\parallel H} = P_{\parallel H}\, x = (I - P_{\perp H})\, x = (I - P_{\parallel v})\, x = \left(I - \frac{v v^*}{\|v\|^2}\right) x = x - \underbrace{\frac{1}{\|v\|^2} (v^* x)\, v}_{x_{\parallel v} = x_{\perp H}}$

reflecting: $x - 2 x_{\perp H} = x - 2 x_{\parallel v} = \underbrace{x - \frac{2}{\|v\|^2} (v^* x)\, v}_{\text{perform } F^{(j)}x \text{ implicitly using } v} = F^{(j)} x \longleftarrow F^{(j)} = I - 2 \frac{v v^*}{v^* v}$

- need: $F^{(j)} L^{(j-1)} = \underbrace{L^{(j-1)} - \frac{2}{\|v^{(j)}\|^2} v^{(j)} \left(v^{(j)*} L^{(j-1)} \right)}_{\text{implicitly multiplies by } F^{(j)}}$

• **check:** is $F^{(j)}$ unitary: let $v = \frac{1}{\|v\|} v$

0) $F^{(j)*} = (I - 2vv^*)^* = I^* - 2v^{**}v^* = I - 2vv^* = F^{(j)}$ (Hermitian: thus $Q^{(j)} = Q^{(j)*}$)

1) $F^{(j)}$ is square $(m-(j-1) \times m-(j-1))$ ✓

2) $F^{(j)*}F^{(j)} = \underbrace{(I - 2vv^*)}_{F^* = F}(I - 2vv^*) = I - 4vv^* + 4v\underbrace{v^*v}_{=\|v\|^2=1}v^* = I - 4vv^* + 4vv^* = I$ ✓

• ok, so, we are almost there...

• how to reflect x to $\overset{|z|=1}{\downarrow}{z}\,\|x\| e_1$?

use the same formula:

$$v = z\|x\|e_1 - x$$

$$F^{(j)}x = x - 2\frac{1}{\|v\|^2}(v^*x)v \,, \quad F^{(j)} = I - 2\frac{vv^*}{\|v\|^2}$$

• what z is best?

• we want to perform the steps:

1. $v = z\|x\|e_1 - x$ ⟶ unstable if v small (cancellation errors)
2. $v = \frac{v}{\|v\|}$ ⟶ unstable if v small (divide by small #)
3. $F^{(j)}x = x - 2(v^*x)v$

so, choose z to maximize $\|v\| \mapsto z = -\mathrm{sign}(x_1)$.

• then: $v = -(\mathrm{sign}(x_1)\|x\|e_1 + x)$

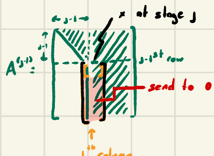
equivalently ($F^{(j)}$ always uses v^*v or vv^*)

$v = \mathrm{sign}(x_1)\|x\|e_1 + x$

• So, Householder **explicitly:**

1. set $A^{(0)} = A$
2. for $j=1$ to n
   (i) $x = A^{(j-1)}_{j:m,\,j}$
   (ii) $v^{(j)} = \mathrm{sign}(x_1)\|x\|e_1 + x$
   (iii) $v^{(j)} = \frac{1}{\|v^{(j)}\|}v^{(j)}$
   (iv) $F^{(j)} = I - 2v^{(j)}v^{(j)*}$
   (v) $Q^{(j)} = \begin{bmatrix} I_{j-1,j-1} & 0 \\ 0 & F^{(j)} \end{bmatrix}$
   (vi) $A^{(j)} = Q^{(j)}A^{(j-1)}$
3. set $R = A^{(n)}$, $Q^* = Q^{(n)}Q^{(n-1)}\dots Q^{(2)}Q^{(1)}$.

reflection, perform implicitly



• **implicitly:** (never build Q or $Q^{(j)}$ or $F^{(j)}$)

1. set $A^{(0)} = A$
2. for $j=1$ to n
   (i) $x = A^{(j-1)}_{j:m,\,j}$
   (ii) $v^{(j)} = \mathrm{sign}(x_1)\|x\|e_1 + x$
   (iii) $v^{(j)} = \frac{1}{\|v^{(j)}\|}v^{(j)}$
   (iv) $A^{(j)}_{(j:m),(j:n)} = A^{(j-1)}_{(j:m),(j:n)} - 2v^{(j)}\left(v^{(j)*}A^{(j-1)}_{(j:m)(j:n)}\right)$
3. set $R = A^{(n)}$, return $\{v^{(j)}\}_{j=1}^{n}$

encode Q implicitly.

- **Householder vs. GS:** $A \in \mathbb{C}^{m \times n}$

pros (H > GS, mGS)

Cost:
1. implicit Householder is faster ($\mathcal{O}(2mn^2 - \frac{2}{3}n^3)$ vs. $\mathcal{O}(2mn^2)$)
2. store Q efficiently via $\{v^{(j)}\}_{j=1}^{n}$

   (is applying Q implicitly cheaper than applying Q explicitly?)

Stability: Householder is provably backward stable

more stable than GS or mGS

cons (GS, mGS > H)

GS & mGS build Q explicitly, in an "online" fashion (each new col $a_j \to q_j$)

Householder builds q_j using all of A, cannot perform sequentially.

Thursday - 4/13/2023

- Logistics:
  - HW 2 due tonight
  - HW 3

- Goals: fast transforms:
  - Ex: building a fast wavelet transform
  - the Fast Fourier Transform !!!

- Fast Transforms: linear transform $T(x) = Ax$, $A \in \mathbb{C}^{m \times n}$ has cost $\mathcal{O}(\overset{m \text{ outputs}}{m}\overset{n \text{ inputs}}{n})$
  - can be prohibitively expensive: image processing - 1080p image has $\approx 2 \times 10^6$ pixels, 3 color values $\longmapsto$ n on order 10^7, $\mathcal{O}(n^2) \approx 10^{14}$ calculations

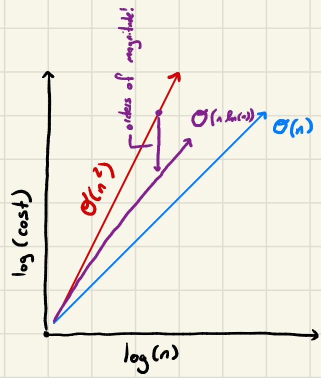
    audio - 44.1 KHz, 10^4 values per second, 3 min $\to 10^6$ values

    $\mathcal{O}(n^2) \approx 10^{12}$ calculations

    video - 24 frames a second, 2 hr movie = 120 min $\times$ 60 s $\times$ 24 $\approx 10^4$

    10^{10} values !! $\mathcal{O}(n^2) \approx 10^{20}$

- yet, we perform signal processing on a massive scale all the time... how?



- idea: use transforms $T(x)$ whose cost is almost $\mathcal{O}(n)$...

- fast transforms: $T(x): \mathbb{R}^n \to \mathbb{R}^n$ cost $\ll \mathcal{O}(n^2)$

  (usually, $\mathcal{O}(n \log_2(n))$)

  way faster, takes $\frac{\log_2(n)}{n}$ less time

  - how?
    - use $\perp$ matrices so inversion is multiplication by A^T, cut cost $\mathcal{O}(n^3) \to$ cost of $A^T x$ } and... stable
    - exploit symmetries of the transform that make certain inner products redundant. } recursive block operations (combine many smaller calculations)

- Ex: $a_1 = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$, $a_2 = \begin{bmatrix} a \\ b \\ -c \\ -d \\ -c \end{bmatrix}$, want $A^T x$, naively: $A^T x$ = 2 inner prod, 9 operations each $\to$ 18 operations

  if we compute $u = \underbrace{[a, b] \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{3 \text{ op}}$, $w = \underbrace{[c, d, e] \cdot \begin{bmatrix} x_3 \\ x_4 \\ x_5 \end{bmatrix}}_{5 \text{ op}}$

  then $A^T x = \begin{bmatrix} u + w \\ u - w \end{bmatrix}$ } 2 op, 10 total

  } 10 operations!

- **Challenge:** design a basis $\{a_j\}_{j=1}^n$ for $\mathbb{R}^n$ s.t.

useful ⟶ 1. the transform $T(y) = Ay$ is meaningful

stable/invertible ⟶ 2. the a's are $\perp$

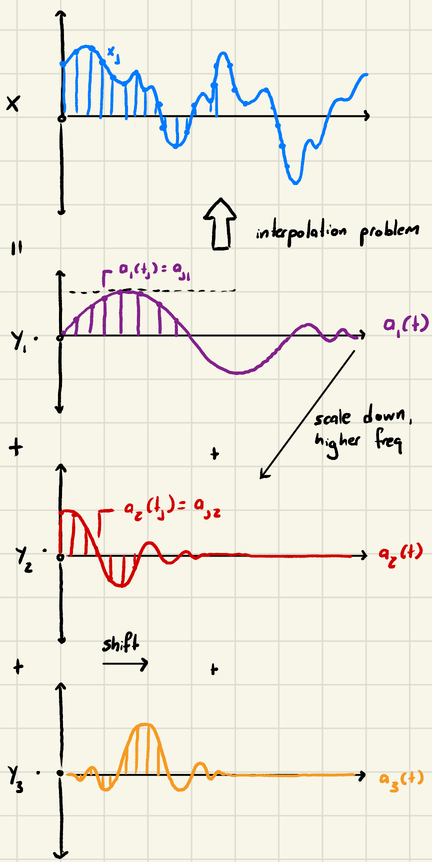
cheap ⟶ 3. the a's are sufficiently symmetric/repetitive...

- **Ex:** a wavelet transform

  x is a signal (say audio)

  x_j = amplitude at time $t_j = j\,\Delta t$

- represent signal as sum of wave(lets) w/ varying frequency & duration:] extract intensity of different pitches at each time



interpolation:

$$A = \begin{bmatrix} a_1(t_1) & a_2(t_1) & \cdots & a_n(t_1) \\ a_1(t_2) & a_2(t_2) & \cdots & a_n(t_2) \\ \vdots & \vdots & & \\ a_1(t_n) & a_2(t_n) & \cdots & a_n(t_n) \end{bmatrix}$$

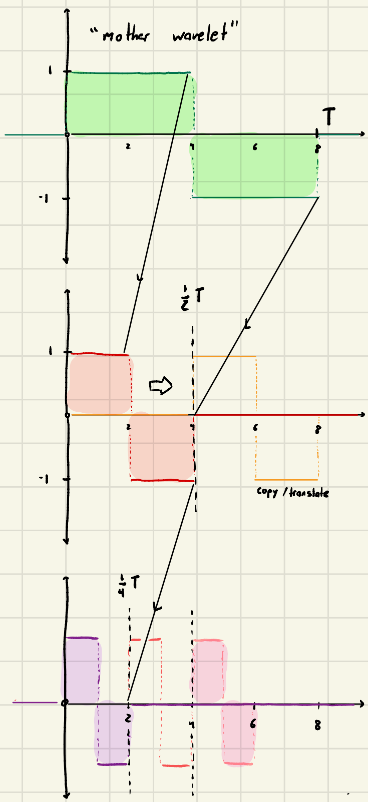
- then A is a "Vandermonde" matrix for functions $\{a_j(t)\}_{j=1}^n$ at samples $\{t_j\}_{j=1}^n$

- solve $Ay = x$ to write

$$x(t) \approx \sum_{j=1}^{n} y_j\, a_j(t)$$

↳ coefficients/representation in "frequency" space

• a particularly easy transform:

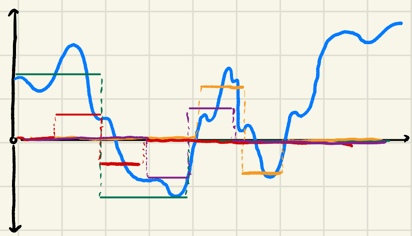


• let $a_1(t)$ be a square wave (one period), T

• $a_2(t) = a_1(2t)$ (scale down)
$a_3(t) = a_1(2(t-T))$ (translate)] 2 copies, half the size

• $a_4(t) = a_2(2t)$
$a_5(t) = a_2(2(t-T))$
$a_6(t) = a_2(2(t-2T))$
$a_7(t) = a_2(2(t-3T))$] 4 copies, $1/4$ the size

• these are wavelets. Call a_j, w_j, A, W
  • represent $x(t)$ as linear comb. of square waves of different frequencies & offsets



• then: $$W^{(m)} = \begin{bmatrix} 1 & 1 & 1 & & 1 & & & \\ 1 & 1 & 1 & & -1 & & & \\ 1 & 1 & -1 & & & 1 & & \\ 1 & 1 & -1 & & & -1 & & \\ 1 & -1 & & 1 & & & 1 & \\ 1 & -1 & & 1 & & & -1 & \\ 1 & -1 & & -1 & & & & 1 \\ 1 & -1 & & -1 & & & & -1 \end{bmatrix}$$

• is $W^{(m)}$ orthogonal? (yes)

• is $W^{(m)}$ cheap to apply? $W^{(m)^T}$ ... ?

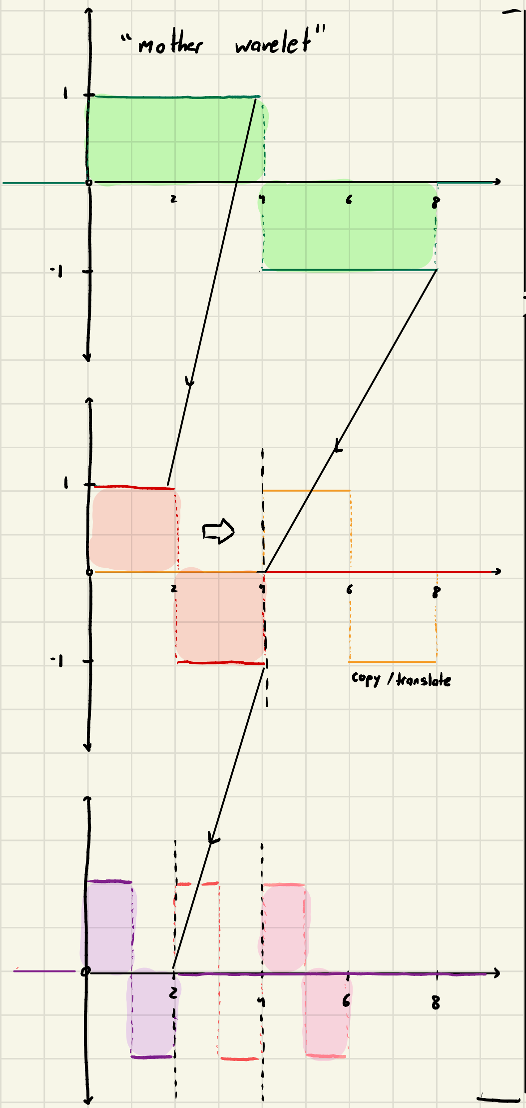
$$
W^{(3)^T} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left[\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & & & & \\
 & & & & 1 & 1 & -1 & -1 \\
1 & -1 & & & & & & \\
 & & 1 & -1 & & & & \\
 & & & & 1 & -1 & & \\
 & & & & & & 1 & -1
\end{array}\right]
\end{array}
$$

← "mother wavelet"

compress by factor of 2

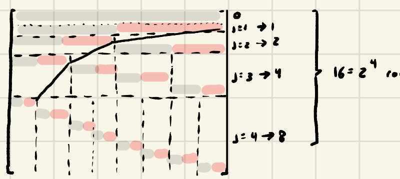
2 spatial intervals

subdivide and compress again



- Ex of a wavelet basis
- build basis by recursively compressing an initial wavelet, then considering all of it's translations

- W for m blocks of size $1, 2^{j-1}$ for $j = 1$ to m is $2^m \times 2^m$

$W^{(4)^T} =$ 

- properties:

1. $\perp$: the col's of $W^{(m)}$ are mutually $\perp$
2. W is square so W is invertible and $W^{-1} \propto W^T$ up to a diagonal scaling (normalize rows)
3. normalization: col 0 has length $\sqrt{\sum_{k=1}^{2^m} 1^2} = \sqrt{2^m}$
$= 2^{m/2}$

col 1 has length $= 2^{m/2}$

col's $j = 2 \dots m$ have $(2^m)/2^{j-1}$ nonzeros

length $= 2^{(m-(j-1))/2}$

so $D_{[(1/2)^{m/2}, (1/2)^{m/2}, (1/2)^{m-1/2}, (1/2)^{m-1/2}, \dots, (1/2)^{1/2}]}$ normalizes

W...

$W^{(m)} D^{(m)}$ is unitary

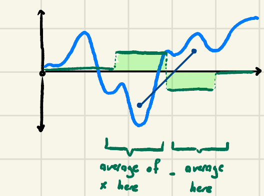
• so $W^{(m)^{-1}} = \text{diag}\big(\underbrace{(\tfrac{1}{2})^m, (\tfrac{1}{2})^m, (\tfrac{1}{2})^{m-1}, (\tfrac{1}{2})^{m-1}, \dots}_{\#\text{ nonzero entries in each row}}\big) W^{(m)^T} = D W^{(m)^T}$

• so, given:

$$x(t) \approx \sum_{j=0}^{n=2^m} y_j w_j(t) \Rightarrow W y = x$$

$$y = D W^{(m)^T} x$$

• notice: y_j is ≈ a difference



• Challenge: compute $W^{(m)^T} x$ faster than $O(n^2) = O(2^{2m})$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\left[\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & & & & \\
& & & & 1 & 1 & -1 & -1 \\
1 & -1 & & & & & & \\
& & 1 & -1 & & & & \\
& & & & 1 & -1 & & \\
& & & & & & 1 & -1
\end{array}\right] \end{array}
\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ \\ x_7 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ \\ y_7 \end{bmatrix}$$

$n = 2^m$ outputs, can't get cheaper than $O(n) = O(2^n)$

• notice: $w_0^T x$ looks a lot like $w_1^T x \dots$

try: compute: $y_0 = w_0^T x = \sum_{j=0}^{n-1} x_j = \sum_{j=0}^{2^m-1} x_j$

then, compute: $y_0^{(1/2)} = \sum_{j=0}^{n/2-1} x_j = \sum_{j=0}^{2^{m-1}-1} x_j$

$$y_1 = \sum_{j=0}^{n/2-1} x_j - \sum_{j=n/2}^{n} x_j$$

$$= y_0^{(1/2)} - (y_0 - y_0^{(1/2)})$$

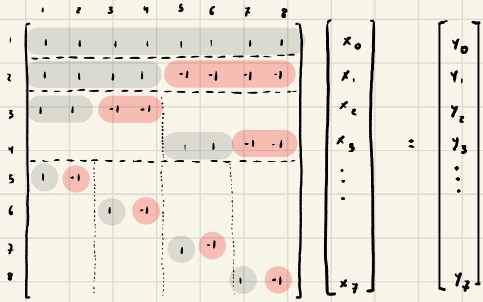
$$= 2y_0^{(1/2)} - y_0$$

Cost: (multiplications)	(additions)		(total)
			to $O(\cdot)$
$n = 2^m$	$n-1 = 2^m - 1$	≈	$2n = 2^{m+1}$
$n/2 = 2^{m-1}$	$n/2 - 1 = 2^{m-1} - 1$	≈	$2\frac{n}{2} = 2^m$
+ 1	1	≈	0

computes y_0 & y_1 in $2(n + \frac{1}{2}n)$ steps
instead of $2(n + n)$ steps
halved the cost for y_1

• to get to $n \log_2(n)$ complexity we recurse:

- y_2 is to $y_1^{(1/2)}$ as y_1 is to $y_0^{(1/2)}$
- if we compute only the inner products w/ the grey shaded entries $(W_j^{(m)} = 1)$, then we can compute $W^{(m)^T} x \dots$



$$\mapsto y_0 = y_0^{(1/2)} + y_0^{(2/2)} \qquad \mapsto y_0$$
$$\mapsto y_1 = y_0^{(1/2)} - y_0^{(2/2)} = 2\,y_0^{(1/2)} - y_0 \mapsto y_0^{(1/2)}$$
$$\mapsto y_2 = y_0^{(1/4)} - y_0^{(2/4)} = 2\,y_0^{(1/4)} - y_0^{(1/2)} \mapsto y_0^{(1/4)}$$
$$\mapsto y_3 = y_0^{(3/4)} - y_0^{(4/4)} = 2\,y_0^{(3/4)} - y_0^{(2/2)} \mapsto y_0^{(3/4)}$$

	cost		cost	
y_0	$2n$		.	$2n$
$y_0^{(1/2)}$	n	$y_0^{(2/2)}, y_0$	2	$n+2$
$y_0^{(1/4)}$	$\frac{1}{2}n$	$y_0^{(2/4)}, y_2$	2	$n+4$
$y_0^{(3/4)}$	$\frac{1}{2}n$	$y_0^{(4/4)}, y_3$	2	
$y_0^{(1/8)}$	$\frac{1}{4}n$	$y_0^{(2/8)}, y_4$	2	$n+8$
$y_0^{(3/8)}$	$\frac{1}{4}n$	$y_0^{(4/8)}, y_5$	2	
$y_0^{(5/8)}$	$\frac{1}{4}n$	$y_0^{(6/8)}, y_6$	2	
$y_0^{(7/8)}$	$\frac{1}{4}n$	$y_0^{(8/8)}, y_7$	2	

$$\text{total cost} = 2\cdot\underbrace{(\text{area shaded grey})}_{n \,+\, (\#\text{ blocks})\cdot n/2} + (2\cdot\#\text{ rows} - 2)$$

$$\#\text{ blocks} = m = \log_2(n)$$

$$\text{total cost} = 2\,(n + \log_2(n)\,n/2) + 2n - 2$$
$$= \log_2(n)\,n + \mathcal{O}(n)$$

a few combinations — many small calculations

$$\text{so} \quad \text{cost} = \mathcal{O}(n\log_2(n))\,!$$

- <u>this is a "fast" transform</u>:

$$\text{cost of} \quad W^{(m)^{-1}} \text{ directly} = \mathcal{O}(n^3) = \mathcal{O}(2^{3m})$$
$$W^{(m)^T} \text{ directly} = \mathcal{O}(n^2) = \mathcal{O}(2^{2m})$$
$$W^{(m)^T} \text{ recursively} = \mathcal{O}(n\log_2(n)) = \mathcal{O}(m\,2^m)$$

- <u>Ex</u>: $m = 20$, $n \approx 10^7$

$= 10^{21}$ ← (ah!)

$= 10^{14}$ ← (ugh...)

$= 10^{8}$ ← hardly more than # of outputs...

asymptotically $\log_2(n)$ calculations per output

<u>savings</u>:

⊥ — 10^7 times faster

symmetry — 10^6 times faster again

- Other fast transforms, matrix multiplication methods use essentially the same trick, recursively block computation to exploit symmetries
  - <u>Ex</u>:
    1. the fast Hadamard-Walsh transform (FHWT)
    2. the fast Fourier transform (FFT)

       (1., 2.) $\mathcal{O}(n\log_2(n))$ instead of $\underbrace{\mathcal{O}(n^2)}_{\text{apply}}$ or $\underbrace{\mathcal{O}(n^3)}_{\text{invert}}$
    3. the Strassen algorithm ($\mathcal{O}(n^{2.807\ldots})$ for AB instead of $\mathcal{O}(n^3)$)

       $\log_2(7)$, not $n^2\log(n)$ since no symmetries assumed.

Gauss, Cooley, Tukey

extremely useful...

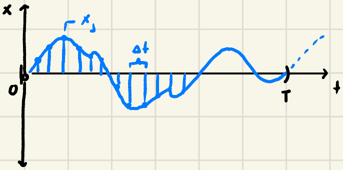
- **The FFT:** performs a fast, discrete Fourier Transform...

  - input: $x = [x_0, x_1, \dots x_{n-1}]$ consider samples of a periodic signal:

    $x_0 = x(0),\ x_1 = x(\Delta t),\ x_2 = (2\Delta t),\ \dots\ x_{n-1} = x((n-1)\Delta t),\ x_n = x(\overbrace{n\Delta t}^{T}) = x(0) = x_0$

    repeats every n samples

    - assume $n = 2^m$, otherwise, zero-pad



  - goal: interpolate on basis of T periodic harmonic functions of increasing frequency

    - e.g.: $\underbrace{\sin(2\pi k \frac{t}{T}),\ \cos(2\pi k \frac{t}{T})}$

      $e^{ik2\pi t/T} = \sin(\dots) + i\cos(\dots)$

      for $k \in [0, 1, 2, \dots n-1]$

    - in terms of the samples:

      $f_k(t) = e^{ik2\pi t/T}$

      $f_k(t_j) = F_{jk} = e^{ik2\pi t_j/T} = e^{ik2\pi \frac{j\Delta t}{n\Delta t}} = (\underbrace{e^{i\frac{2\pi}{n}}}_{\text{"}\omega\text{"}})^{jk} = \omega^{jk}$

so

$$F^{(n)} = \begin{array}{c|c} & \text{increasing freq } k \rightarrow \\ \hline \begin{array}{c} t_0 \\ t_1 \\ \downarrow \\ t_{n-1} \end{array} & \begin{bmatrix} | & | & & | \\ f_0(t) & f_1(t) & \dots & f_{n-1}(t) \\ | & | & & | \end{bmatrix} \end{array}$$

$$= \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \dots & \omega^{n-1} \\ \omega^0 & \omega^2 & \omega^3 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{bmatrix}$$

  - then, Vandermonde matrix of $[\omega^0, \omega^1, \omega^2, \dots \omega^{n-1}]$ is the Fourier (discrete) matrix: $F^{(n)}_{jk} = \omega^{jk}$ for $\omega = e^{i\frac{2\pi}{n}}$

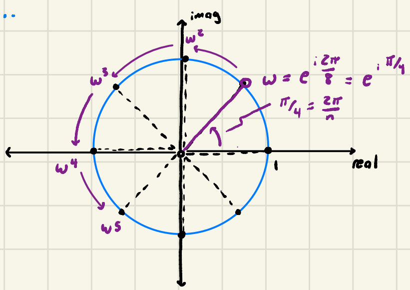
    Fact: $F^{(n)}$ has $\perp$ columns, (not normalized, length $\sqrt{n}$) invert via $F^{(n)*} = \bar{F}^{(n)}$

    - What is ω?

      - rule for complex exponentiation: given $z = |z|e^{i\theta}$, $z^k = |z|^k e^{ik\theta} = |z|^k \overbrace{(\cos(k\theta) + i\sin(k\theta))}^{\text{rotate}}$

      - raising ω^k rotates in complex plane...

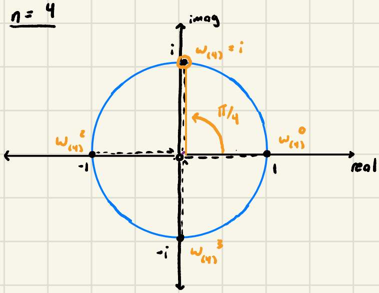
      - $\omega = e^{i\frac{2\pi}{n}}$ = the 1^{st} of the n complex roots of 1...



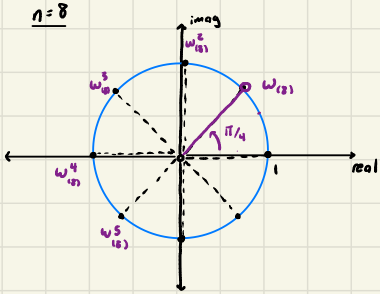
Ex: $n = 8$

compare: $\omega_{(n)}^{\ell}$ for $n = 4, 8$

$n = 4$



$n = 8$



notice: $\omega_{(2n)}^{2\ell} = \omega_n^{\ell}$ $\quad$ (Ex: $\omega_{(8)}^2 = \omega_{(4)}^1 = i$, $\omega_{(8)}^4 = \omega_{(4)}^2 = -1$, etc...)

thus: powers of ω repeat, $F_{(n)}$ and $F_{(2n)}$ share entries... redundancy in a recursive block / multiscale fashion

· Ex: $n = 4$, $\omega = i$

$$F_{(4)} = \begin{array}{c} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} \begin{array}{c} j= \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \\ k = \; 0 \quad 1 \quad 2 \quad 3 \end{array} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

Swap even & odd columns to match entries $\quad$ (use $P = \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$)

$$\begin{array}{c} \text{even} \qquad \text{odd} \\ \left[\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & i & -i \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & -i & i \end{array}\right] \\ k = \; 0 \quad 2 \quad 1 \quad 3 \end{array} = \begin{array}{c} \left[\begin{array}{c|c} A & B \\ \hline A & -B \end{array}\right] \\ \text{even } k \quad \text{odd } k \end{array}$$

← match, only need 1 mult each

← match, only need 1 mult each

blocks are redundant after swapping even & odd

the blocks are related to $F_{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $(\omega = -1)$ $\quad A = F_{(2)}$, $B = \begin{bmatrix} 1 & \\ & i \end{bmatrix} F_{(2)} = D_{(2)} F_{(2)}$

$$\text{so} \quad = \begin{bmatrix} F_{(2)} & D_{(2)} F_{(2)} \\ F_{(2)} & -D_{(2)} F_{(2)} \end{bmatrix} = \begin{bmatrix} I & D_{(2)} \\ I & -D_{(2)} \end{bmatrix} \begin{bmatrix} F_{(2)} & F_{(2)} \\ F_{(2)} & F_{(2)} \end{bmatrix}$$

so:
$$F_{(4)} = \underset{\substack{\uparrow \\ \text{change signs} \\ \text{(sparse)}}}{S_{(2)}} \underbrace{\begin{bmatrix} F_{(2)} & F_{(2)} \\ F_{(2)} & F_{(2)} \end{bmatrix}}_{\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes F_{(2)} = (\mathbb{1}\mathbb{1}^T) \otimes F_{(2)}} \underset{\substack{\uparrow \\ \text{swap even \& odd}}}{P_{\text{even-odd}}},$$

"Kronecker Product"

we can build $F_{(4)}$ from $F_{(2)}$!

- **Generally:** $F_{(n)} = S_{(n/2)} \left((\mathbb{1}\mathbb{1}^T) \otimes F_{(n/2)} \right) P_{\text{even-odd}}$

- **Consequence:** we can perform $\bar{F}_{(n)} x$ using:

  1. permute even & odd, use $P_{(n/2)}$ ← splits $x = \overbrace{[x_0, x_1, x_2, x_3, \dots x_{n-1}]}^{\text{length } 2^m}$ into 2 signals
     $x_{even} = [x_0, x_2, x_4, \dots x_{n-2}]$, $x_{odd} = [x_1, x_3, x_5, \dots x_{n-1}]$ } length $n/2 = 2^{m-1}$
  2. apply $\bar{F}_{(n/2)}$ to x_{even}, x_{odd}
  3. apply a sparse change of signs $S_{(n/2)}$ and combine

- **costs:**
  1. cheap, $\mathcal{O}(n)$ swaps ← linear time
  2. 2 (cost of $F_{(n/2)}$) ⟵ dominates
  3. cheap, $\mathcal{O}(2n)$ nonzero entries ← linear time

- now we can recurse... if $n = 2^m$ then, every time we need to use $\bar{F}_{(\ell)}$ use $\bar{F}_{(\ell/2)}$ twice...

1. apply $\bar{F}_{(2^m)}$ → $2\times$ [1. permute even & odd, use $P_{(2^{m-1})}$; 2. apply $\bar{F}_{(2^{m-1})}$ to x_{even}, x_{odd}; 3. apply a sparse change of signs $S_{(2^{m-1})}$ and combine] → $2\times$ [1. permute even & odd, use $P_{(2^{m-2})}$; 2. apply $\bar{F}_{(2^{m-2})}$ to x_{even}, x_{odd}; 3. apply a sparse change of signs $S_{(2^{m-2})}$ and combine] → $2\times$ [1. permute even & odd, use $P_{(2^{m-3})}$; 2. apply $\bar{F}_{(2^{m-3})}$ to x_{even}, x_{odd}; 3. apply a sparse change of signs $S_{(2^{m-3})}$ and combine]

$1 \times \bar{F}_{(2^m)} = 2 \times \bar{F}_{(2^{m-1})} = 4 \times \bar{F}_{(2^{m-2})} = 8\, \bar{F}_{(2^{m-3})}$

(8 ← # of times → n; $\bar{F}_{(2^{m-3})}$ ← smaller computation)

- so: applying $\bar{F}_{(2^m)}$ = applying $\bar{F}_{(2^{m-\ell})}$ 2^ℓ times $\times\ \ell$ conversions = applying $\bar{F}_{(1)}$ $2^m = n$ times $\times\ m = \log_2(n)$ conversions! ⟵ many small computations ⟵ a few combinations

- cost of FFT is $\mathcal{O}(n \log_2(n))$ if $n = 2^m$!
  ↑ incredibly cheap

basis for:
1. virtually all signal processing – denoising, convolution, filtering, CT scanning...
2. data compression & transfer
3. spectral analysis – think spectroscopy, exoplanet detection, seismology
4. soln methods for PDE's – heat eqn, diffusion, waves, SDE's

Week 5 - Inverse Problems (direct methods)

- Tuesday - 04/18/2023 - Linear Systems

  - **Logistics:**
    - finish reading up to part IV
    - HW 3 assigned Tuesday, due next Tuesday (April 25th)
    - Project 2 post, due May 10th

  - **Goals:**
    - introduction to inverse problems
    - Gaussian Elimination
      - LU decomposition
      - cost
      - stability & pivoting

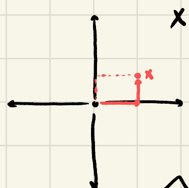
- **Inverse Problems:** given a transform ("forward model"), $T: X \to Y$ and an output $y \in Y$, find x s.t.

$$T(x) = y.$$

- **Ex:** imaging, interpolation, steady state problems for dynamical systems, optimizer updates, etc.

  - if X, Y finite dimensional, T linear, then solving a **linear system**: given $A \in \mathbb{C}^{m \times n}$, $y \in \mathbb{C}^m$ find $x \in \mathbb{C}^n$ s.t.

$$Ax = y$$



  - **general soln:** if T is injective/one-to-one then $T(x) = T(x')$ iff $x = x'$

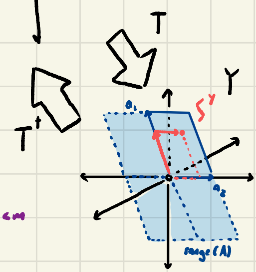
    let $T^{\dagger}: \text{range}_{x \in X}\{T(x)\} \longrightarrow X$ s.t. $T^{\dagger}(y) = x$ if $T(x) = y$

    "pseudo-inverse"

    $y \in \text{range}(A)$

    ... $A^{\dagger} \in \mathbb{C}^{n \times m}$ s.t. $A^{\dagger}y = x$ if $Ax = y \Rightarrow A^{\dagger}Ax = x$

    $A^{\dagger}A = I_{n \times n}$ ← $n \leq m$



  - if T is bijective (injective & surjective) then $\text{range}_x\{T(x)\} = Y$

    so $T^{\dagger} = T^{-1}$, $T^{-1}(y) = x$ s.t. $T(x) = y$ $\forall y \in Y$

    ... A square, full rank $\Rightarrow A^{\dagger} = A^{-1}$, $A^{-1}y = x$ if $Ax = y \Rightarrow A^{-1}A = I_{n \times n} = AA^{-1}$

- **Numerically:** never compute A^{-1}. Why? usually only need x for a couple y's, not all y (cheaper) $\|A^{-1}\| \gg \|A\|$ for many examples, lose structure in A (sparsity), and, only need A^{-1} implicity to compute x... use reduction methods instead.

- **Linear Systems:** given $A \in \mathbb{C}^{m \times n}$, $b \in \mathbb{C}^{m \times 1}$ find $x \in \mathbb{C}^{n \times 1}$ s.t. $Ax = b$

  - Ex: $\left.\begin{array}{l} \text{(i)}\ x + 2y = 3 \\ \text{(ii)}\ 4x + 5y = 6 \end{array}\right\}$ grouping $\left\{\begin{array}{l} \text{unknowns: } x, y \longrightarrow x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{array}{l} = x \\ = y \end{array}, \\ \text{Knowns: } b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix} \end{array}\right.$

  given $A = \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix}$, $b = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$ find x s.t. $Ax = b$.

  - Ex: $\left.\begin{array}{l} \text{(i)}\ 4u + 2v + 2w = 10 \\ \text{(ii)}\ 2u - 3v + 0w = -1 \\ \text{(iii)}\ -2u + 7v + 2w = 9 \end{array}\right\}$ $\begin{bmatrix} 4 & 2 & 2 \\ 2 & -3 & 0 \\ -2 & 7 & 2 \end{bmatrix} \underset{\text{“}x\text{”}}{\begin{bmatrix} u \\ v \\ w \end{bmatrix}} = \begin{bmatrix} 10 \\ -1 \\ 9 \end{bmatrix} \longmapsto \overset{\overbrace{}^{A}\ x = \overbrace{}^{b}}{\left[\begin{array}{ccc|c} 4 & 2 & 2 & 10 \\ 2 & -3 & 0 & -1 \\ -2 & 7 & 2 & 9 \end{array}\right]}$

  augmented matrix

  multiply by x and set equal to

1. **Row Reduction:**
   combine eqn's (rows) to eliminate variables
   i.e. to set entries to zero

$$\begin{array}{c} \text{(i)} \\ \text{(ii)} \\ \text{(iii)} \end{array} \left[\begin{array}{ccc|c} 4 & 2 & 2 & 10 \\ 2 & -3 & 0 & -1 \\ -2 & 7 & 2 & 9 \end{array}\right] \underset{\frac{1}{2}\text{(i)} + \text{(iii)}}{\xrightarrow{-\frac{1}{2}\text{(i)} + \text{(ii)}}} \left[\begin{array}{ccc|c} 4 & 2 & 2 & 10 \\ 0 & -4 & -1 & -6 \\ 0 & 8 & 3 & 14 \end{array}\right] \xrightarrow[2\text{(ii)} + \text{(iii)}]{} \underbrace{\left[\begin{array}{ccc|c} 4 & 2 & 2 & 10 \\ 0 & -4 & -1 & -6 \\ 0 & 0 & 1 & 2 \end{array}\right]}_{\text{upper triangular}}$$

cancel out

2. **Back-Substitution:** now, solve from the bottom up

$$\left[\begin{array}{ccc|c} 4 & 2 & 2 & 10 \\ 0 & -4 & -1 & -6 \\ 0 & 0 & 1 & 2 \end{array}\right] \begin{array}{c} \text{(i)} \\ \text{(ii)} \\ \text{(iii)} \end{array}$$

(iii) $0x_1 + 0x_2 + 1x_3 = 2 \longmapsto x_3 = 2$

(ii) $0x_1 - 4x_2 - 1x_3 = -6 \longmapsto -4x_2 - 2 = -6 \longmapsto -4x_2 = -4 \longmapsto x_2 = 1$

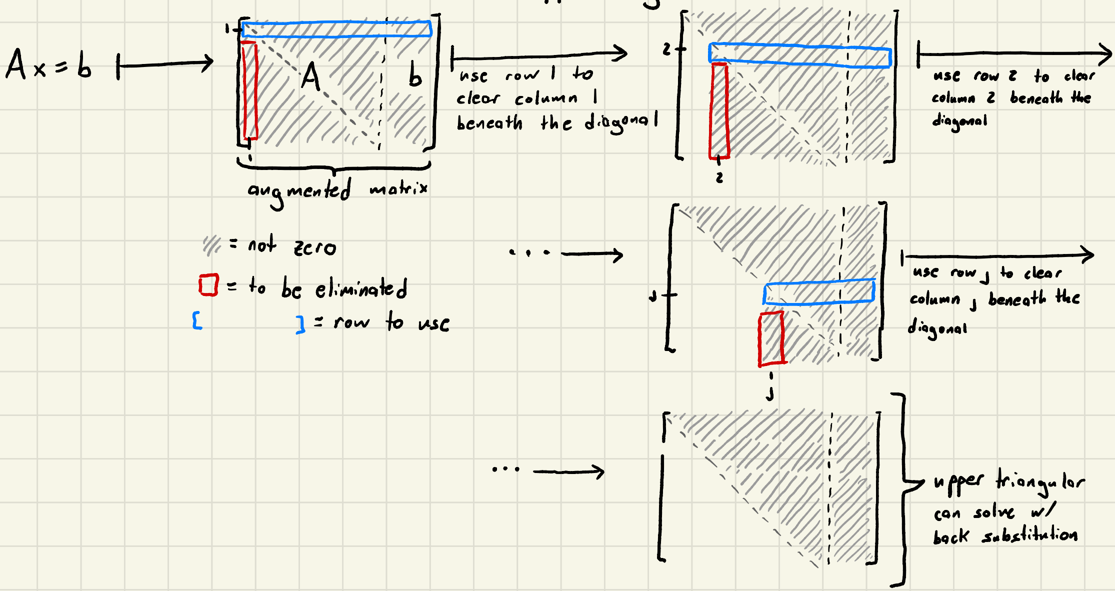
(i) $4x_1 + 2x_2 + 2x_3 = 10 \longmapsto 4x_1 + 2 + 4 = 10 \longmapsto 4x_1 + 6 = 10$

$\longmapsto 4x_1 = 4 \longmapsto x_1 = 1$

- So, $x = [1, 1, 2]$

this method is <u>Gaussian Elimination</u>: general algorithm for solving linear systems

1. <u>row-reduce</u>: use rows to cancel columns until upper triangular

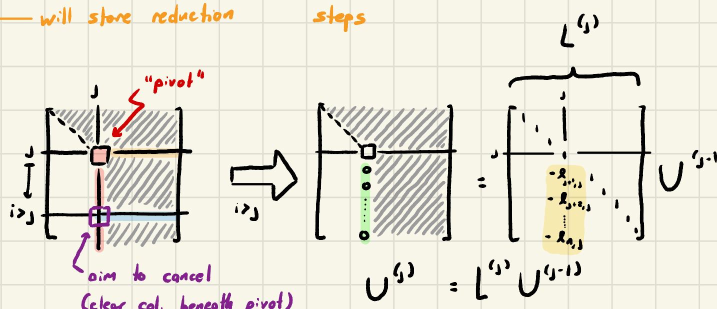


2. <u>back-sub</u>: solve backwards bottom to top.

- <u>Algorithm</u>: input $A \in \mathbb{C}^{n \times n}$, $b \in \mathbb{C}^n$ ← assume A is square, non-singular

Gaussian - Elimination

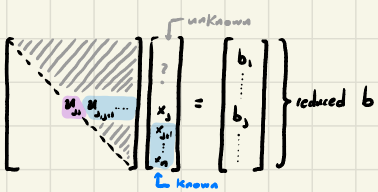
1. initialize $U = A$, $L = I_{n \times n}$ ← will store reduction steps
2. for $j = 1$ to n
   (i) for $i = j+1$ to n
      (a.) $\ell_{ij} = u_{ij} / u_{jj}$
      (b.) $u_{i,j:n} = u_{i,j:n} - \ell_{ij} u_{j,j:n}$
      (c.) $b_i = b_i - \ell_{ij} b_j$ ← apply same row operations to b as A



back-sub.

1. initialize $x = [\,]_{n \times 1}$
2. for $j = n$ to 1
   (i) $x_j = (b_j - \sum_{i>j} u_{ji} x_i) / u_{jj}$

solves upper triangular systems (forward sub if lower triangular)



The LU decomposition:

- **Def:** given $A \in \mathbb{C}^{n\times n}$ invertible, $\exists$ a permutation matrix $P \in \mathbb{R}^{n\times n}$, s.t.

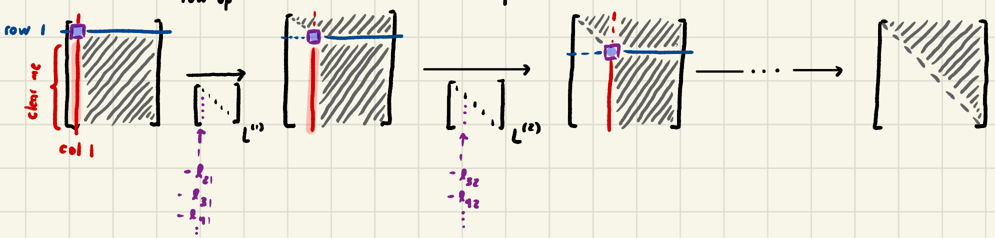
$$\underbrace{P}_{\text{reorder the rows}} A = LU$$

where:

1. $L \in \mathbb{C}^{n\times n}$ lower triangular (w/ $l_{jj} = 1$, $\underbrace{|l_{ij}| \leq 1 \;\; \forall\, i > j}_{\text{goal of permuting improves stability}}$)
2. $U \in \mathbb{C}^{n\times n}$ upper triangular

- **reduction:** Gaussian elimination reduces $A \longrightarrow U$ via

$$A = U^{(0)} \xrightarrow[\text{row op}]{} U^{(1)} = L^{(1)}U^{(0)} \xrightarrow[\text{row op}]{} U^{(2)} = L^{(2)}U^{(1)} \longrightarrow \cdots \longrightarrow U = L^{(n-1)}U^{(n-2)}$$



- **so:** $\overbrace{L^{(n-1)}L^{(n-2)}\cdots L^{(2)}\underbrace{L^{(1)}A}_{U^{(1)}}}^{L^{-1} = (L^{(n-1)}L^{(n-2)}\cdots L^{(2)}L^{(1)})} = U \Rightarrow \overset{\text{"does" reduction}}{L^{-1}}A = U$ so $A = \overset{\text{"undoes" reduction}}{L}U$

$\underbrace{L^{(2)}U^{(1)}}_{U^{(2)}}$

$\vdots$

$$L^{-1} = \begin{bmatrix} 1 & & & & \\ -l_{21} & 1 & & & \\ -l_{31} & -l_{32} & 1 & & \\ \vdots & & & \ddots & \\ -l_{n1} & -l_{n2} & & & 1 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & & & \ddots & \\ l_{n1} & l_{n2} & & & 1 \end{bmatrix}$$

reduction: l's are the multipliers used

$$u^{(j)}_{i,j:n} = u^{(j-1)}_{i,j:n} - l_{ij}\, u^{(j-1)}_{j,j:n}$$

$L^{-1} \Longleftrightarrow L$ by negating entries below the diagonal.

- build L implicitly during reduction

- advantage of LU: given $A \in \mathbb{C}^{n \times n}$, L, U known

solve $Ax = b \overset{L(Ux)=b}{\Longleftrightarrow}$ 1. solve $Ly = b$ via forward sub
2. solve $Ux = y$ via back sub

- much faster if L, U are known, or solving $Ax=b$ for a sequence of b's...

- <u>Computational Cost</u>: how expensive is Gaussian Elimination ($A \to LU$)?
Back-Sub?

same as QR...

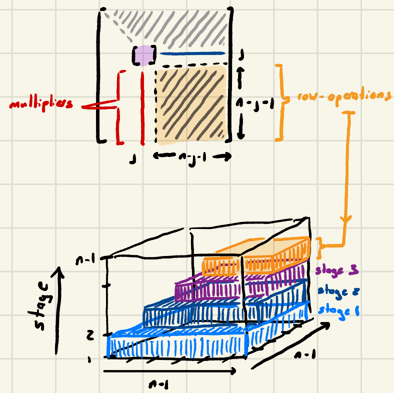
- <u>Row-reduction</u>: given A $n \times n$, the cost of red. is $\propto n^3$ ($\mathcal{O}(n^3)$)

expensive/slow

- Why? 1. $n-1$ stages (j from 1 to $n-1$)
2. operate on $n-j-1$ rows w/ $n-j$ col's, only need for $n-j-1$ col's
(cost: 1 multiplier per row

row op = $1 \otimes + 1 \ominus$ per entry = $\underset{\substack{\uparrow \\ \#\text{ rows}}}{(n-j-1)}(1 + 2(n-j-1))$

$= 2(n-j-1)^2 + \mathcal{O}(n-j-1))$



$$\text{cost} = \sum_{j=1}^{n-1} 2((n-1)-j)^2 = \frac{2}{3}n^3 + \mathcal{O}(n^2) \sim \frac{2}{3}n^3$$

to leading order

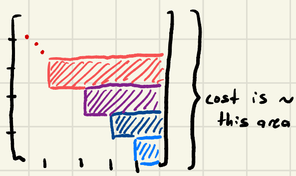
- <u>Back-Sub</u>: cost $\propto n^2$ ($\mathcal{O}(n^2)$)

(n stages, at j^{th} stage do $\sim 2(j-1)+1$ operations,

$$\text{cost} \sim \sum_{j=1}^{n} 2(j-1) = 2\left(\frac{(n+1)n}{2} + n\right)$$

$$\sim n^2 + \mathcal{O}(n))$$

so, $\text{cost} \sim n^2$



- <u>moral</u>: row-reduction is much more expensive than back-sub.

$\mathcal{O}(n^3)$ vs. $\mathcal{O}(n^2)$

- general trend:
  - reduction/factorization is expensive: $\mathcal{O}(n^3)$ ← same cost as multiplication!
  - using reduced/factorized matrices is cheap: $\mathcal{O}(n^2)$ or faster

... n^3 is cost to reduce, n^2 is cost to use...

- given $A = F_\ell F_{\ell-1} \dots F_1$ for some factors

if we want Ax could

$$\left\{\begin{array}{l} \text{compute } A = (F_1 F_2 \dots F_\ell) \Rightarrow Ax \\ \quad \text{cost: } \mathcal{O}((\ell-1)n^3), \ \mathcal{O}(n^2) \end{array}\right\} \text{explicit, slow}$$

$$\left\{\begin{array}{l} \text{compute: } x^{(j)} = F^{(j)} x^{(j-1)}, \ x^{(0)} = x \\ \quad \text{cost: } \mathcal{O}((\ell-1)n^2) \end{array}\right\} \text{implicit, fast}$$

don't multiply factors, apply in sequence

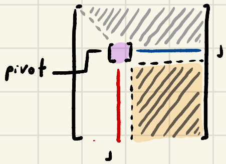
$$Ax = F_\ell(F_{\ell-1}(\dots F_2(F_1 x))\dots))$$

work w/ decompositions implicitly when possible

- Stability & Pivoting: is Gaussian elimination $A \to LU$ stable?

no!, not w/out pivoting (lecture 21)

- Suppose that, at stage j:



the pivot $u_{jj}^{(j-1)}$ is very small

(or $u_{jj}^{(j-1)} \ll u_{ij}^{(j-1)}$ for $i > j$)

then multiplier $\ell_{ij} = u_{ij}^{(j-1)} / u_{jj}^{(j-1)}$ requires division by a small number!

unstable (extreme case: $u_{jj}^{(j-1)} = 0$, divide by 0 error!)

- in principle there is no need to reduce in order...

- Ex:

$$\begin{bmatrix} 1 & 3 & -1 \\ 0 & \boxed{0} & 2 \\ 0 & 4 & -6 \end{bmatrix} x = \begin{bmatrix} -2 \\ 4 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} 1 & 3 & -1 \\ 0 & 4 & -6 \\ 0 & 0 & 2 \end{bmatrix} x = \begin{bmatrix} -2 \\ 3 \\ 4 \end{bmatrix}$$

ah! pivot = 0

has the same soln if we reorder the rows

- idea: pivoting: at each stage, $\underbrace{\text{reorder rows}}_{\text{adds } \mathcal{O}((n-j)^2) \text{ search cost... too expensive}}$ & columns of $U^{(j-1)}$ to maximize the pivot

partial pivoting: only swap rows ← only adds $\mathcal{O}((n-j))$ search cost.

• **LU w/ partial pivoting algorithm**: at each stage, swap rows to maximize the pivot

input $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$

1. initialize $U = A$, $L = I_{n\times n}$, $P = I_{n\times n}$ ← stores the swaps
2. for $j = 1$ to n
   - (i) find $K = \underset{i\geq j}{\operatorname{argmax}} \{|u_{ij}|\}$ ← cost $\mathcal{O}(n-j)$ ↑ # entries to search
   - (ii) pivot:
     - (a) $u_{j,j:n} \rightleftarrows u_{K,j:n}$, $b_j \rightleftarrows b_K$
     - (b) $\ell_{j,1:j-1} \rightleftarrows \ell_{K,1:j-1}$
     - (c) $p_{j,:} \rightleftarrows p_{K,:}$

     (b)–(c): book-keeping

     swap rows j & K of U, b, L, P

     cost $\mathcal{O}(n-j)$ ↑ # entries per row
   - (iii) for $i = j+1$ to n
     - (a.) $\ell_{ij} = u_{ij}/u_{jj}$
     - (b.) $u_{i,j:n} = u_{i,j:n} - \ell_{ij}\, u_{j,j:n}$
     - (c.) $b_i = b_i - \ell_{ij}\, b_j$

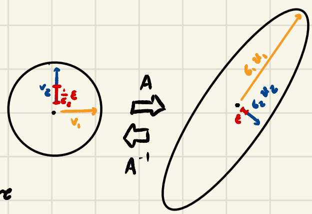
     reduce as before

     cost $\mathcal{O}((n-j)^2)$ still dominates, asymptotic cost unchanged.

then: $PA = LU$

• **Question**: is Gaussian Elimination + pivoting ⟶ substitution stable? backward stable? (lectures 17 & 22)

1. **Conditioning**: given $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$ the worst-case conditioning of the problem: find $x = A^{-1}b$ is $\kappa(A) = \|A\|\,\|A^{-1}\|$ ($= \sigma_{max}/\sigma_{min}$ if use $\|\cdot\|_2$) - recall conditioning of linear transforms lecture
   - ill-conditioned if $\sigma_{min} \ll \sigma_{max}$, $\|A^{-1}\| \gg \|A\|$



2. **Stability of Back (or forward) Sub**:

   Thm: given $U \in \mathbb{C}^{n\times n}$ upper triangular, then solving for x s.t. $Ux = b$ via substitution is backward stable, i.e. the computed soln $\tilde{x}$ satisfies:
   $$(U + \delta U)\tilde{x} = b$$
   where: $|\delta u_{ij}|/|u_{ij}| \leq n\,\varepsilon_m + \mathcal{O}(\varepsilon_m^2)$ $\forall\, i \leq j$ $\therefore$ $\|\delta U\|/\|U\| = \mathcal{O}(\varepsilon_m)$.

   solve almost the right problem exactly!

3. What about Gaussian Elimination: $A \to P, L, U$?? (Lecture 22)

- Gaussian Elimination is <u>explosively</u> unstable for some pathological examples but... in practice is always* stable... acts as if backward stable for real problems.

  (* for the class of matrices appearing in real problems)

- First, Gaussian Elimination + pivoting is neither stable nor backward stable
  - in the following we assume $A = PA$ (pivot to optimal order a priori)

  - <u>Thm</u>: given $A = LU \in \mathbb{C}^{n \times n}$, compute $\tilde{L}, \tilde{U}$ via Gaussian Elimination

    then:

    $$\underbrace{\tilde{L}\tilde{U} = A + \delta A}_{}, \quad \|\delta A\| = \|L\|\,\|U\|\,\mathcal{O}(\varepsilon_m)$$

    backward stability... like QR, $\tilde{L} \not\approx L$ and $\tilde{U} \not\approx U$

    only hope $\tilde{L}\tilde{U} \simeq A$...

  - <u>problem</u>: w/ pivoting, $|l_{ij}| \le 1$ w/ $=$ iff $i=j$ so $\|L\| = \mathcal{O}(1)$ ← limit large n

    in any norm, so

    $$\|\delta A\| = \|U\|\,\mathcal{O}(\varepsilon_m)$$

    but, $\|U\|$ can be $\gg \|A\|$ so the relative error

    $$\|\delta A\|/\|A\| = \underbrace{\|U\|/\|A\|}_{\text{large...}}\,\mathcal{O}(\varepsilon_m) \text{ can be large}$$

    - <u>define</u>: the growth factor $\rho = \dfrac{\max_{ij}|u_{ij}|}{\max_{ij}|a_{ij}|}$. then $\|U\| = \mathcal{O}(\rho\,\|A\|)$

  - <u>Thm</u>: given $PA = LU$, $A \in \mathbb{C}^{n \times n}$ computed via Gauss. Elim. w/ partial pivoting then $\tilde{P} = P$ for sufficiently small ε_m if $|l_{ij}| < 1$ for all $i > j$ (no ties in pivoting choice) and

    $$\tilde{L}\tilde{U} = \tilde{P}A + \delta A, \quad \text{where: } \frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\rho\,\varepsilon_m)$$

    - then, backward stable if $\rho = \mathcal{O}(1)$ in n...

      but, consider:

      $$A^{(4)} = \begin{bmatrix} 1 & & & & 1 \\ -1 & 1 & & & 1 \\ -1 & -1 & 1 & & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix} \text{ then } U = \begin{bmatrix} 1 & & & & 1 \\ & 1 & & & 2 \\ & & 1 & & 4 \\ & & & 1 & 8 \\ & & & & 16 \end{bmatrix} \text{ so } \rho = 16 = 2^{n-1}$$

  - <u>Fact</u>: $\rho \le 2^{n-1}$ and $\exists$ a sequence of $\{A^{(n)}\}_{n=1}^{\infty}$, $A^{(n)} \in \mathbb{C}^{n \times n}$ s.t. $\rho = 2^{n-1}$...
  - looks like a disaster, lose $\mathcal{O}(n)$ digits of accuracy for linear systems size $n \times n$!!!

  - in practice: extremely rare. $\rho \ll 2^{n-1}$ ← test this! propose an experiment...

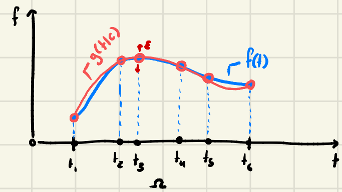
Thursday - 04/20/2023 - Least Squares Problems

- **Logistics:**
  - HW 3 posted, due next Tuesday

- **Goals:**
  - Least Squares Problems:
    - motivation - dangers of exact soln's
    - definition $ geometry
    - the normal eqn's vs. projection
    - direct methods:
      1. Projection $ QR
      2. Normal eqn's $ Cholesky
      3. Psuedo-inversion $ SVD

- **Motivation - Approximate Soln's $ Fitting:**

  - **Ex:** Consider the interpolation problem:
    1. sample times $\{t_j\}_{j=1}^{m} \in \Omega$
    2. signal: $f(t) \to \{f_j\}_{j=1}^{m}$, $f_j = f(t_j) + \varepsilon$



try to interpolate w/ function basis $\{b_k\}_{k=1}^{n}$, $b_j : \Omega \to \mathbb{R}$

  - **Ex:** $b_1(t) = 1$, $b_2(t) = t$, ... $b_k(t) = t^{k-1}$ (poly. interp.)

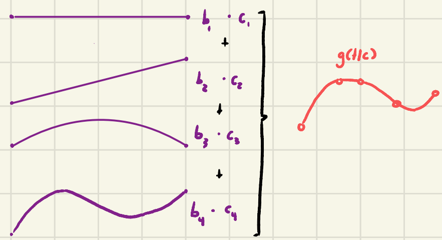
want coefficient vector $\vec{c}$ s.t.

$g(t|c) = \sum_{k=1}^{n} c_k b_k(t)$ interpolates f at the samples

$$g(t_j|c) = \sum_{k=1}^{n} c_k \underbrace{b_k(t_j)}_{B_{jk}} = f(t_j) = f_j$$

where:

$$B = \underbrace{\begin{bmatrix} b_1(t_1) & b_2(t_1) & \cdots & b_n(t_1) \\ b_1(t_2) & b_2(t_2) & \cdots & b_n(t_2) \\ \vdots & \vdots & & \vdots \\ b_1(t_m) & b_2(t_m) & \cdots & b_n(t_m) \end{bmatrix}}_{n} \Big\updownarrow m$$



so, solve: $\underbrace{\begin{bmatrix} B \end{bmatrix}}_{m \times n} \begin{bmatrix} c \end{bmatrix}_{n} = \begin{bmatrix} f \end{bmatrix}_{m}$

- **Problem:** an interpolant $g(t|c)$ only exists if $f \in \text{range}(B)$ and is unique iff B is full rank ($m \geq n$, lin. ind. col's)

  - two ideas:
    1. add basis functions until $f \in \text{range}(B) \Rightarrow m = n \Rightarrow$ for polynomials, m data points $\Rightarrow$ degree $m-1$ polynomial

       numerically very risky... over fitting! unstable!
    2. give up on exact soln's (interpolation) and approximate instead (fit)

• Least Squares Problems... Solving $Ax \approx b$

• **Suppose:** given A $m \times n$ and b $m \times 1$
usually seek x s.t. $Ax = b$
what if $b \notin \text{range}(A)$?
then no x exists s.t. $Ax = b$

} often occurs when overconstrained...
more eqn than unknowns
$m > n$

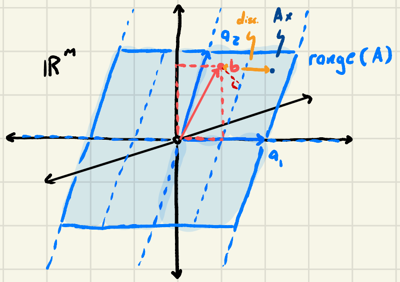
this is the standard setting when fitting data to a model:

data = Ax + measurement error

so data $\notin$ range of A

or,

data $\approx Ax$ but w/ modeling error



so, no x exists s.t. $Ax = b$, let's find x s.t. Ax is as close to b as possible, that is x that minimizes the discrepancy:

discrepancy $= Ax - b$ want $Ax \approx b$ so

LS problem: find x which minimizes $\|Ax - b\|_2$
equivalent to:
find x which minimizes $\|Ax - b\|_2^2 = \sum_{i=1}^{m} \underbrace{(Ax - b)_i}_{\text{disc. in } i^{th} \text{ entry}}^2$

= sum of squares of discrepancy.

• **Fact:** the LS problem has a soln for all A & b ← a soln always exists
(note: the soln may not be unique)

• **What about uniqueness?** suppose x_* to the LS problem and $\exists$
$z \in \text{null}(A)$, $z \neq 0$

$\|A(x_* + z) - b\|^2 = \|Ax_* + \underbrace{Az}_{0} - b\|^2 = \|Ax_* - b\|^2$

then $x_* + z$ is also a soln, so soln's are not unique.

• **Fact:** the LS problem has a unique soln iff A is full rank
(linearly independent columns $\Rightarrow m \geq n$)

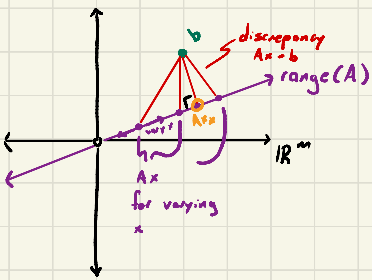
- LS problems are the <u>most</u> widely solved opt. problems...
  1. data is often noisy, noise ~ Gaussian dist, prob $\propto \exp(-\|\text{discrepancy}\|^2)$
  2. LS problems are "easy" to solve at large scale
  3. LS problems admit many methods/approaches
  4. LS problems are easily generalized, adaptible

- ok, so how do we solve them?

- <u>A geometric soln for LS problems</u>:

  - given A $m\times n$, b $m\times 1$ find: $x_* = \underset{x\in\mathbb{C}^n}{\text{argmin}}\left(\|Ax-b\|^2\right)$

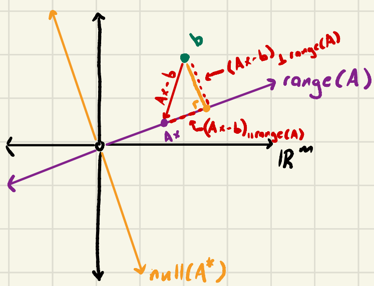
  - a picture to build intuition...

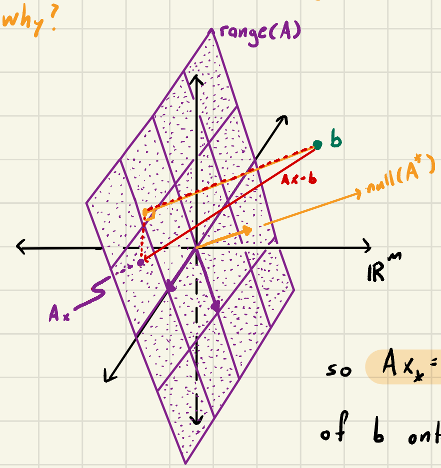


- try a series of x values

- <u>notice</u>: x_* is chosen such that the discrepancy is $\perp$ to the range(A).

- why?





b projected onto range(A)

so $Ax_* = b_{\parallel \text{range}(A)}$ = $\perp$ proj of b onto range(A)

- $b\in\mathbb{R}^m$ so $Ax-b\in\mathbb{R}^m$ $\therefore$ can decompose the discrepancy into a component in range(A) and null(A*)... (Fund. Thm: $\mathbb{R}^m = \text{range}(A) \oplus \text{null}(A^*)$)

$$Ax-b = (Ax-b)_{\parallel \text{range}(A)} + (Ax-b)_{\parallel \text{null}(A^*)}$$ ← $= \perp$ range(A)

- moreover $(Ax-b)_{\| range(A)}$ is $\perp$ $(Ax-b)_{\| null(A^T)}$ (Fund. Thm. part II)

  so, by pythagorus:

$$(*) \quad \|Ax-b\|^2 = \underbrace{\|(Ax-b)_{\| range(A)}\|^2}_{\text{depends on } x} + \underbrace{\|(Ax-b)_{\| null(A^*)}\|^2}_{\text{same for all } x}$$

so, minimizing $\|Ax-b\|^2$ is the same as minimizing $\|(Ax-b)_{\| range(A)}\|^2$

$$(**) \quad b_{\| range(A)} \in range(A) \text{ so } \exists \text{ an } x_* \text{ s.t. } (Ax_*-b)_{\| range(A)} = 0$$

$\therefore$ by $(*)$ $\|Ax-b\|^2 \geq \|(Ax-b)_{\| null(A^*)}\|^2$ for all x

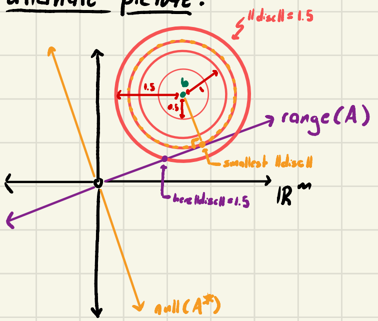
and by $(**)$ at x_* s.t. $(Ax-b)_{\| range(A)} = 0$, $\|Ax-b\|^2 = \|(Ax-b)_{\| null(A^*)}\|^2$

so if x_* solves the LS problem then $(Ax_*-b)_{\| range(A)} = 0 \ldots$

$$\text{discrepancy} = Ax_*-b = (Ax_*-b)_{\| null(A^*)} + 0 \in null(A^*)$$

$\therefore$ discrepancy is $\perp$ range(A), discrepancy $\in null(A^*)$

- <u>alternate</u> <u>picture</u>:



at soln x_* the discrepancy must be $\perp$ to range(A), since it is a radial vector to the circle/sphere centered at b of smallest radius intersecting range(A). The circle must be tangent to the range there, so the radial vector is $\perp$ the range.

$\therefore$ discrepancy $\in null(A^*)$

- <u>Consequence</u>: at soln x_*, discrepancy $\in null(A^*)$

  so...

$$A^*(\text{discrepancy}) = A^*(Ax_*-b) = 0$$

rearrange:

$$\underbrace{(A^*A)x_* = A^*b}_{\text{the normal eqn's}} \longleftarrow \text{but this is a linear system!!}$$

- Can solve all LS problems by solving a linear system!

• **Thm:** given $A \in \mathbb{C}^{m\times n}$, $m \geq n$ w/ linearly ind. col's (full rank) and $b \in \mathbb{C}^m$ then the LS problem:

$$\text{find } x_* = \underset{x \in \mathbb{C}^n}{\arg\min} \{ \|Ax - b\|^2 \}$$

has a unique soln for any b, and x_* is the unique soln to the $n\times n$ linear system:

$$\text{(normal eqn's)} \quad (A^* A)\, x_* = A^* b .$$

if $m \geq n$ then "overconstrained", more constraints than degrees of freedom:

$$\underbrace{\left[\, A^* \,\right]}_{n \times m} \; \underbrace{\left[\, A \,\right]}_{m \times n} = \underbrace{\left[\, A^*A \,\right]}_{n\times n}, \qquad \underbrace{\left[\, A^* \,\right]}_{n \times m} \; \underbrace{\left[\, b \,\right]}_{m} = \underbrace{\left[\, A^*b \,\right]}_{n}$$

$(n\times m) \times (m\times n) = (n\times n)$, $\quad (n \times m) \times (m) = (n)$

$$\text{then: } \left[\, A^*A \,\right] \left[\, x \,\right] = \left[\, A^*b \,\right]$$

$(n\times n) \times (n) = (n)$

• the normal eqn's compress the problem to an $n\times n$ system of equations!

• <u>notice</u>: if A $m\times n$, full rank then A^*A is $n\times n$, full rank (lin. ind. col's) — try proving this!

so $(A^*A)^{-1}$ exists...

$$x_* = \underbrace{(A^*A)^{-1} A^*}_{\text{psuedo-inverse}} b = A^\dagger b$$

• <u>Def</u>: given $A \in \mathbb{C}^{m\times n}$, $m \geq n$, full rank, the <u>pseudo-inverse</u> of A is

$$A^\dagger = (A^*A)^{-1} A^*$$

and $A^\dagger b = x_*$ solves the LS problem, minimize $\|Ax - b\|^2$ $\forall\, b \in \mathbb{C}^m$

• how to compute $A^+ = (A^*A)^{-1}A^*$?

use a decomp to avoid explicit inverses...

• Ex (SVD): $A = U\Sigma V^* \Rightarrow A^* = V\Sigma^* U^*$

$$A^*A = V\Sigma^*U^*U\Sigma V^* = V\underbrace{\Sigma^*\Sigma}_{(n\times n)}V^*$$

$$\underbrace{\begin{bmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n\end{bmatrix}}_{(n\times m)}\begin{bmatrix}\sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & & & \sigma_n \\ & & & \end{bmatrix} = \underbrace{\begin{bmatrix}\sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_n^2\end{bmatrix}}_{\longleftarrow n \longrightarrow} = \operatorname{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2) = D_\sigma^2$$

$$(A^*A)^{-1} = V D_\sigma^{-2} V^*$$

$$(A^*A)^{-1}A^* = V D_\sigma^{-2}\underbrace{V^*V}_{=I}\Sigma^*U^* = V\left(D_\sigma^{-2}\begin{bmatrix}\sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \sigma_n \\ & 0 & & \end{bmatrix}\right)U^* = V\overbrace{\begin{bmatrix}1/\sigma_1 & & & \\ & 1/\sigma_2 & & \\ & & \ddots & 1/\sigma_n \\ & 0 & & \end{bmatrix}}^{\text{reciprocate all } \sigma \neq 0}U^*$$

so,

$$A^+ = V\Sigma^+U^* = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix}\begin{bmatrix}1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_n\end{bmatrix}\begin{bmatrix} — & u_1^* & — \\ & \vdots & \\ — & u_n^* & — \end{bmatrix}$$

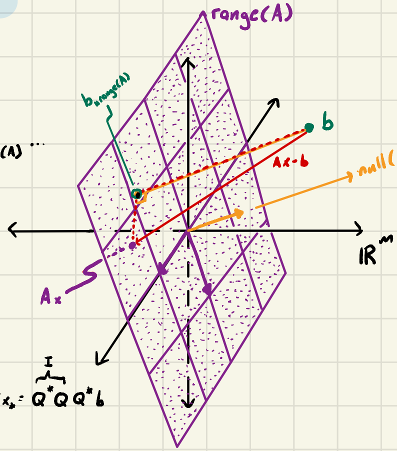
• So, applying A^+b implicitly:

1. $A = U\Sigma V^*$ 2. $w = \begin{bmatrix} — u_1^* — \\ \vdots \\ — u_n^* — \end{bmatrix} b$ 3. $y_j = \frac{1}{\sigma_j} w_j$ 4. $x_* = Vy$.

• alternatively: discrepancy $= (Ax_* - b) \in null(A^*) = range(A)^\perp$

$$\therefore (Ax_* - b)_{\|range(A)} = 0 \quad \text{so} \quad \underbrace{(Ax_*)}_{\text{already} \in range(A)}{}_{\|range(A)} = b_{\|range(A)} \dots$$

$$Ax_* = b_{\|range(A)}$$ ← a linear system!



• Soln: given A $m\times n$, full rank

1. $A = QR$, $Q_{m\times n}$, $R_{n\times n}$

then $(QR)x_* = P_{range(A)}b = (QQ^*)b \Rightarrow$ multiply by Q^*... $\overbrace{Q^*Q}^{I}Q^*Rx_* = \overbrace{Q^*Q}^{I}Q^*b$

2. solve $Rx_* = Q^*b$ w/ substitution

↑ notice: same as plugging $A = QR$ into A^+...

$$x_* = A^+b = (A^*A)^{-1}A^*b = (R^*\underbrace{Q^*Q}_{=I}R)^{-1}R^*Q^*b = (R^*R)^{-1}R^*Q^*b$$

$$= R^{-1}\underbrace{R^{*-1}R^*}_{=I}Q^*b = R^{-1}Q^*b \quad \text{so} \quad x_* = R^{-1}Q^*b \Rightarrow Rx_* = Q^*b.$$

• gives three different direct methods...

↑ decomp based, solve exactly in exact arithmetic in finitely many steps

	Stability		Cost
1. solve w/ QR (projection based)	stable enough	(backward stable)	in-between (usually, fast enough)
2. solve w/ SVD (psuedo-inverse based)	most stable	(backward stable)	slowest
3. solve w/ normal equations (LU of A^*A → Cholesky)	unstable!!	(squares $\kappa(A)$ when a good fit is available!!!)	fastest

↑ like LU, but exploits symmetry

- Comparison of direct methods:

the Goldilocks method, compromise stability & speed,
* numerical method of choice (usually)

1. Projection: use QR.

a) $A = QR$ (Householder) ← backward stable, cost: $\mathcal{O}(2mn^2 - \frac{2}{3}n^3)$

b) compute $Q^* b$ (implicit) ← cond. = 1, backward stable ($m \geq n$), cost: $\mathcal{O}(mn)$ ← naive

c) solve $R x_* = Q^* b$ (substitution) ← backward stable, cost: $\mathcal{O}(n^2)$

cost ~ $2mn^2 - \frac{2}{3}n^3$

- Stability: backward stable, $\|(A + \delta A)\tilde{x} - b\| = \|A x_* - b\|$, $\frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_m)$

conditioning = conditioning of original LS problem

- Cost: ~ $2mn^2 - \frac{2}{3}n^3$ (cost of QR)

2. Psuedo-Inverse: use SVD

a) $A = U \Sigma V^*$ (we don't know how to do this yet) ← ??, cost $\mathcal{O}(2mn^2 + 11n^3)$

b) $y_j = \frac{1}{\sigma_j}(u_j^* b)$ ← cond = $\kappa(A)$, backward stable ($n \leq m$), cost $\mathcal{O}(mn)$

c) $x_* = Vy$ ← cond = 1, backward stable, cost $\mathcal{O}(n^2)$

cost ~ $2mn^2 + 11n^3$

- Stability: backward stable, $\|(A + \delta A)\tilde{x} - b\| = \|A x_* - b\|$, $\frac{\|\delta A\|}{\|A\|} = \mathcal{O}(\varepsilon_m)$

conditioning = conditioning of original LS problem

- Cost: ~ $2mn^2 + 11n^3$ (cost of SVD... roughly)

3. Normal Equations: use Cholesky (LU for A^*A) * most naive implementations BE CAREFUL

a) $y = A^* b$ ← backward stable ($n \leq m$), cond = $\kappa(A)$, cost $\mathcal{O}(mn)$

b) $M = A^* A$ ← backward stable ($n \leq m$), but... squares conditioning, cost $\mathcal{O}(mn^2)$

c) $M = LU$ ← unstable, backward stable in practice, cost $\mathcal{O}(\frac{2}{3}n^3)$

d) solve $Lw = y$ ($w = L^{-1}y$, implicit via reduction) ← backward stable, cond. = 1 w/ pivoting, cost $\mathcal{O}(n^2)$

e) solve $U x_* = w$ (back-sub) ← backward stable, cost $\mathcal{O}(n^2)$

cost ~ $mn^2 + \frac{2}{3}n^3$

- Cholesky replaces step c) w/

c') $M = R^* R$ (R $n \times n$, upper triangular)

d') solve $R^* w = y$ ($y = R^{*^{-1}} y$, implicit via reduction)

e') solve $R x_* = w$ (back-sub)

all backwards stable
cost of Cholesky = $\frac{1}{3}n^3$
(half the cost, exploits symmetry of M)

- Stability: Unstable. Squares the conditioning when $\kappa(A)$ large, or, when a good fit (small discrep soln) is available!!!

- Cost: ~ $mn^2 + \frac{1}{3}n^3$ ($\frac{1}{2}$ cost w/ QR)

- <u>The Cholesky Decomposition</u>: symmetrized LU (see lecture 23)

    - Given $M \in \mathbb{C}^{m \times m}$ (square) that is
        1. Hermitian: $M^* = M$
        2. positive-definite: $x^* M x > 0 \quad \forall\, x \neq 0$

      then $\exists\ R \in \mathbb{C}^{n \times n}$, upper triangular s.t.

$$\begin{bmatrix} & & \\ & M & \\ & & \end{bmatrix} = \begin{bmatrix} \ddots & & \\ & R^* & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \ddots & & \\ & R & \\ & & \ddots \end{bmatrix}$$

like LU
except "L" = "U*"

- <u>Fact</u>: if $M = A^* A$, $A \in \mathbb{C}^{m \times n}$ full rank then M is Hermitian and positive definite.

- We will continue this story in HW 4...

- this completes our study of direct methods
  going forward will use <u>iterative/optimization methods</u>.

Week 6 - Iterative Methods for LS problems (and intro to optimization)

- Logistics:
  - Project 1 due Wednesday
  - HW 3 due Thursday
  - HW 4 posts tonight, due next Thursday

- Goals:
  - LS as an optimization problem
  - general survey of optimization & system solving
  - general survey of methods
  - convergence rate analysis
  - Ex:
    - gradient descent for least squares
    - Convergence rates, conditioning and scaling
    - accelerating gradient descent (momentum & accelerated GD)
  - Preview to: Sloppy but cheap (SGD) vs. careful but expensive (CGLS)

- Least Squares as Optimization:

  - given $A \in \mathbb{C}^{m \times n}$, $m \geq n$, full rank, $b \in \mathbb{C}^{m \times 1}$ goal find $\underbrace{x \in \mathbb{C}^n}_{\text{domain is all of } \mathbb{C}^n}$ minimizing:

$$f(x) = \|Ax - b\|_2$$

$$\Downarrow$$

$$f(x) = \|Ax - b\|_2^2 = \sum_{i=1}^{m} ([Ax]_i - b_i)^2$$

$[Ax]_i = (i^{th} \text{ row of } A) \cdot x$

  - looking for x_*:

$$x_* = \underset{x \in \mathbb{C}^n}{\arg\min} \{ f(x) \}$$

where $f : \mathbb{C}^n \to \mathbb{R}$,

$$f(x) = \|Ax - b\|^2$$

$([Ax]_i - b_i)^2$ error in the approx:

$$(i^{th} \text{ row of } A) \cdot x \approx b_i$$

- **Optimization more generally:**

find $x_* = \underset{x \in \Omega}{\arg\min} \{ f(x) \}$, $f: \Omega \to \mathbb{R}$

- domain: Ω · objective: f

- nonlinear system: $h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_m(x) \end{bmatrix}$, $h_i: \Omega \to \mathbb{C}$

solve $h(x) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ $\leftarrow$ linear system: $h_i(x) = [Ax]_i - b_i$

generally, there are not direct approaches for solving

$h(x) = 0$

in arbitrary dimensions if h are nonlinear...

- Ex: suppose $h_i(x)$ is a polynomial degree > 1

at simplest h_i is quadratic

- if degree $= 2$, solving $h(x) = 0$ is equivalent to finding the roots of a degree $2m$ polynomial (if $m > 3$) impossible $2m > 4$.

- convert into optimization problem: $f(x) = \|h(x)\|$

- Ex: $f(x) = \|h(x)\|_p = \left[\sum_{i=1}^{m} (|h_i(x)|^p) \right]^{1/p}$

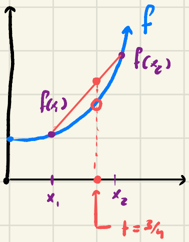
- LS: h_i are linear, $p = 2$.

- **Classifying Optimization:**

  - domain Ω: 1. is a pointset $\Rightarrow$ discrete/combinatorial opt. (CS)
    2. $\Omega \subset \mathbb{R}^n$, $\Omega \neq \mathbb{R}^n \Rightarrow$ constrained opt. problems
    3. $\Omega = \mathbb{R}^n \Rightarrow$ unconstrained problem $\leftarrow$ LS problems

• objective:

1. convexity of f: f is convex over a set S if for any $x_1, x_2 \in S$, and any $t \in [0,1]$



$$(1-t)f(x_1) + t f(x_2) \geq f((1-t)x_1 + t x_2)$$

[strictly convex $>$ for $t \in (0,1)$

• more generally: if f is convex over S

then f[convex comb. of points in S] $\leq$ convex comb. of f[points]

[strictly $<$

• typically use convexity to guarantee existence & uniqueness of minima

• Ex: if Ω is compact, f is strictly convex then $\exists$ a unique minimizer x_* (global)

• Ex: if f is bounded below, strictly convex $\Rightarrow$ if $\exists$ a minimizer x_* then it is unique ← LS problems when A is full rank.

• f is locally convex, if f is convex over a set $S \subset \Omega$

• f is nonconvex if it is not globally convex

[typically hard, admit many local minima...

2. smoothness of f: how differentiable is f?

• what derivatives exist & where

• how large are high order derivatives

] key to controlling the predictiveness of local information

control viable step sizes

• LS problems have quadratic f: all derivatives exist but they may differ in scale...

• how to extend local information... local models, Taylor series...

• Ex: expand $f(x)$ about some iterate x_k:

$$f(x) \approx f(x_k) + \nabla_x f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k)$$

- where $\nabla_x f(x_k) = \begin{bmatrix} \partial_{x_1} f(x_k) \\ \vdots \\ \partial_{x_n} f(x_k) \end{bmatrix}$, $H(x_k) = \begin{bmatrix} \partial_{x_1}^2 & \partial_{x_1}\partial_{x_2} & \cdots \\ \partial_{x_2}\partial_{x_1} & \partial_{x_2}^2 & \\ \vdots & & \ddots \end{bmatrix} f(x) \Big|_{x = x_k}$

$$f(x) \approx \underbrace{f(x_k) + \nabla_x f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k)}_{\text{local quadratic model} \Rightarrow \text{takes the same form as } f \text{ for a LS problem.}} + \mathcal{O}(\|x - x_k\|^3)$$

- Optimality criteria:
  - Ex: if x_* ∈ interior of Ω, f is convex on a neighborhood S containing x_* then x_* is a local minimizer if $\nabla_x f(x_*) = 0$ ← LS problem. } 1st order optimality criteria
  - Ex: if problem is constrained ⇒ different criteria (Lagrange multipliers, KKT conditions)

- Methods (iterative) for convex optimization:
  - idea: sequence of guesses to the soln x_* that get better with each update

$$\{x_k\}_{k=0} : x_0 \to x_1 \to x_2 \to \cdots \approx x_*$$

  - typically update $x_k \to x_{k+1}$ using $f(x_k)$, $\approx \nabla f(x_k)$, $\approx H(x_k) \ldots$
    use:
    1. history of iterates
    2. history of local estimates

    ⇒ leads to methods employing "momentum" or "acceleration"

  - typically: $x_{k+1} = x_k + s_k z_k$, $\Omega \subseteq \mathbb{R}^n$
    - where $s_k \in \mathbb{R}$ (step size)
    - where $z_k \in \mathbb{R}^n$ (step direction)

    } choosing these strategically

    - def: a direction z_k is a descent direction if, for all sufficiently small s

$$\underbrace{f(x_k + s z_k) \leq f(x_k)}_{\text{descent step.}}$$

- order of a method is the degree of derivatives used to compute z_k:
  1. if we only use $\approx \nabla_x f(x_k)$ $\Rightarrow$ 1^{st} order (Gradient descent) ← requires $\mathcal{O}(n)$ differences
  2. if ... $\approx \nabla_x f(x_k)$, $\approx H(x_k)$ $\Rightarrow$ 2^{nd} order (Newton or quasi-Newton) ← requires $\mathcal{O}(n^2)$ differences

  - tradeoff: 2^{nd} order methods are faster per step but steps are more expensive & less-robust (require more controls)

- we can speed computation of update z_k by using cheaper local approximations:

  1. limiting the # of variables x_j that can change per update $\Rightarrow$ coordinate descent } sub-sample col's of $Ax \approx b$
  2. or, if
  $$f(x) = \sum_{i=1}^{m} f_i(x) \leftarrow f(x) = \|h(x)\|_p^p = \sum_{i=1}^{m} |h_i(x)|^p$$
  update using subsets of f_i at each step } sub-sample rows of $Ax \approx b$

  - in either approach ordering of the subsets chosen controls convergence rate
  - often better to perform blockwise in a random or stochastic pattern $\Rightarrow$ Stochastic Gradient Descent.

    - for LS: choose sampling of rows & col's based on their relative norms
      - Stochastic Gradient Descent for LS.

- Convergence Rates: rate at which error $\|x_k - x_*\|$ goes to zero

  - typically: statements like, $\exists$ a constant $C > 0$ s.t.
  $$\|x_{k+1} - x_*\| \le C\,\|x_k - x_*\| \Rightarrow \text{errors decay geometrically}$$
  $$\|x_k - x_*\| \le C^k\,\|x_0 - x_*\|$$
  } most iterative methods for solving LS problems

  convergence is controlled by this constant C.

  for LS, depends on method used and the conditioning of the matrix A

  - def: a sequence $\{x_k\}_{k=0}^{\cdots}$ converges linearly if $\|x_k - x_*\| = \mathcal{O}(\|x_{k-1} - x_*\|)$ ← first order methods

    quadratically if $\|x_k - x_*\| = \mathcal{O}(\|x_{k-1} - x_*\|^2)$, etc. ← higher order methods

Thursday - 04/27/2023 - Iterative Methods for LS

- Logistics:
  - HW 4 posted, due next Thursday
- Goals:
  - The LS objective & quadratic objectives
  - Gradient Descent:
    - fixed step size
    - exact line search
  - Momentum & Acceleration for ill-conditioned problems

- Last Class: surveyed optimization problems - Today: focus on LS

- Least Squares: given $A \in \mathbb{C}^{m \times n}$, full rank, $b \in \mathbb{C}^m$ find $x_* \in \mathbb{C}^n$ s.t.

$$x_* = \underset{x \in \mathbb{C}^n}{\operatorname{argmin}} \{ \|Ax - b\|^2 \}$$

- domain: $\Omega = \mathbb{C}^n$
- objective: $f : \Omega \to \mathbb{R}$, $f(x) = \frac{1}{2}\|Ax - b\|^2 = \frac{1}{2}(Ax-b)^*(Ax-b)$

$$= \tfrac{1}{2}(x^* A^* A x - x^* A^* b - b^* A x + b^* b)$$

$$= \tfrac{1}{2}(x^* A^* A x - 2(A^* b)^* x + \underbrace{b^* b}_{\|b^2\|})$$

$M = A^* A$ is Hermitian positive definite ($x^* M x > 0 \ \forall x \neq 0$)

so f is convex

- view as special case of:
  - general quadratic (convex) objective:

$$f(x) = \tfrac{1}{2}(x^* M x + 2y^* x + c), \quad M \in \mathbb{C}^{n \times n} \text{ Hermitian p.d.}$$
$$y \in \mathbb{C}^n, \quad c \in \mathbb{R}$$

  - Ex: if $f: \Omega \to \mathbb{R}$, convex at x_*, analytic:

$$f(x) \simeq f(x_*) + (\nabla_x f(x_*))^T (x - x_*) + \tfrac{1}{2}(x - x_*)^T H(x_*)(x - x_*) + \mathcal{O}(\|x - x_*\|^3)$$

$c/2$

$+y, = 0$ if x_* is local minimizer

Hessian: $\begin{bmatrix} \partial^2_{11} & \partial^2_{12} & \cdots & \partial^2_{1n} \\ \partial^2_{21} & \partial^2_{22} & & \vdots \\ \partial^2_{n1} & \cdots & & \partial^2_{nn} \end{bmatrix} f(x) \Big|_{x = x_*}$ M

Hermitian, positive definite if convex at x_*

- Moral: study optimization on LS objective to study generic behavior of optimizers near local minima, or, when using a local quadratic model $\leftarrow$ controls steps for 2nd order methods

- can also view as special case of system solving via optimization:

$$h : \Omega \to \mathbb{C}^m, \quad h_i : \Omega \to \mathbb{C}, \quad h(x) = \begin{bmatrix} h_1(x) \\ \vdots \\ \vdots \\ h_m(x) \end{bmatrix}$$

aim to solve:

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_m(x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0$$

so, minimize residual:

$$\|h(x)\|_p^{\,p} = \sum_i |h_i(x)|^p = \begin{matrix} |h_1(x)|^p \\ + \\ |h_2(x)|^p \\ + \\ \vdots \\ |h_m(x)|^p \end{matrix} = \begin{matrix} |h_1(x) - 0|^p \\ + \\ |h_2(x) - 0|^p \\ + \\ \vdots \\ |h_m(x) - 0|^p \end{matrix}$$

for LS: $h(x) = Ax - b$, $p = 2$

$$h(x) = \begin{bmatrix} [Ax]_1 - b_1 \\ [Ax]_2 - b_2 \\ \vdots \\ [Ax]_m - b_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0$$

so, minimize residual:

$$\|h(x)\|^p = \|Ax - b\|^2 = \sum_{i=1}^{m} |[Ax]_i - b_i|^2 = \begin{matrix} |[Ax]_1 - b_1|^2 \\ + \\ |[Ax]_2 - b_2|^2 \\ + \\ \vdots \\ |[Ax]_m - b_m|^2 \end{matrix}$$

- choose p to control regularity of $f(x) = \|h(x)\|_p^{\,p}$ and the sense in which errors are small ($p \to \infty$, max error, $p \to 0$, # of constraints unsatisfied)

- when motivated by regression: depends on the error model in data ($p = 2 \Leftrightarrow$ Gaussian errors)

- <u>batching</u>: only focus on subsets of rows/constraints or subsets of variables at a time...

- <u>batch</u>: only use variables x_j for $j \in S$, constraints $i \in C$

$$\text{minimize} \quad \sum_{i \in C} ([Ax]_i - b_i)^2 \quad \text{over} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \quad \text{w/ } x_j \text{ fixed unless } j \in S$$

  - <u>Ex</u>: $S = \{1, 2, 4\}$, $C = \{2, 3\}$.

$$\begin{array}{c|c} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} \boxed{} & & \boxed{} & \\ & & & & \\ \boxed{} & & A & \boxed{} & \\ & & & & \\ & & & & \end{bmatrix} \\ & \begin{matrix} \uparrow & \uparrow & & \uparrow & \\ \in & S & & \in S & \end{matrix} \end{array} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \approx \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} \begin{matrix} \leftarrow \in C \\ \\ \leftarrow \in C \\ \\ \\ \end{matrix} \Bigg\} \text{new smaller LS problem}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{14} \\ a_{31} & a_{32} & a_{34} \end{bmatrix} \overset{x_S}{\overset{\downarrow}{\begin{bmatrix} x_1 \\ x_2 \\ x_4 \end{bmatrix}}} \approx \begin{bmatrix} b_1 \\ b_3 \end{bmatrix} - \begin{bmatrix} a_{13} & a_{15} \\ a_{33} & a_{35} \end{bmatrix} \overset{x_{S^c}}{\overset{\downarrow}{\begin{bmatrix} x_3 \\ x_5 \end{bmatrix}}}$$

- Solving batched problems is the same as solving a LS problem sizes $m = |C|$, $n = |S|$ for small $|C|$, $|S|$ use direct methods (analytic answers available for $|C| = 1$ or $|S| = 1$)
- cost $\mathcal{O}(2|C||S|^2 - \tfrac{2}{3}|S|^3)$ per batch if direct, otherwise $\longrightarrow$ back to iterative LS methods
- pick batch order/sets stochastically $\Rightarrow$ stochastic gradient methods (convergence rate depends on sampling rate, usually not uniform over rows/col's)

• Shape of quadratic (LS) objectives:

- given $f(x) = \frac{1}{2} x^* M x + y^* x + \overbrace{\tfrac{1}{2} c}^{\text{doesn't change position of optimizer}}$

  LS: $M = A^*A$, $y^* = -A^*b$, $c = b^*b$

  Taylor: $M = H(x_*)$, $y^* = \nabla f(x_*)$, $c = 2f(x_*)$

  $\Big\}$ for $M \in \mathbb{C}^{n\times n}$: Hermitian, positive-definite, invertible

- minimized at x_* then: $\nabla f(x_*) = 0$

$$\nabla f(x) = Mx + y \quad \text{so} \quad \nabla f(x)\big|_{x=x_*} = Mx_* + y = 0 \longmapsto Mx_* = -y \longmapsto x_* = -M^{-1}y$$

$$\text{LS: } M = A^*A,\ y = -A^*b \longmapsto x_* = -M^{-1}y = \underbrace{(A^*A)^{-1}A^*}_{\text{psuedo-inverse}} b \ \checkmark$$

simplify by shifting to be centered at x_* ...

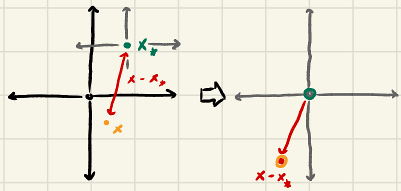
$$\frac{1}{2}(x - x_*)^* M (x - x_*) = \frac{1}{2} x^* M x - \underbrace{(Mx_*)^*}_{Mx_* = -y} x + \underbrace{\frac{1}{2} x_*^* M x_*}_{\text{constant } d} = \frac{1}{2} x^* M x + y^* x + d$$

$$= f(x) + \text{some constant...}$$

so:

$$f(x) = \frac{1}{2}(x - x_*)^* M (x - x_*) \text{ up to additive constant}$$

WLOG, assume $x_* = 0$ (else, shift coordinate system)



- M is Hermitian so it is unitarily diagonalizable, has real eigenvalues $\lambda_j(M)$ and $\perp$ eigenvectors $v_j(M)$ s.t.

$$M = V \Lambda(M) V^* \qquad V^* = V^{-1}$$

and is positive definite so $\lambda_j(M) > 0 \ \forall j \in [1, n]$.

LS: $M = A^*A$, if A has SVD $A = U\Sigma V^*$ then: $A^*A = V\Sigma^*\Sigma V^*$

so $\Lambda(M) = \Sigma^*(A)\Sigma(A)$

... $\lambda_j(M) = \sigma_j(A)^2 > 0$

- then $f(x) = \frac{1}{2} x^* M x = \frac{1}{2} x^* V \Lambda(M) V^* x$

$$= \frac{1}{2}(V^*x)^* \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} (V^*x)$$

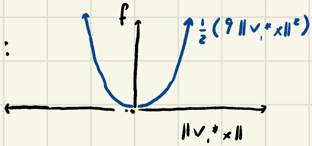
$$= \sum_{j=1}^{n} \frac{1}{2} \lambda_j(M) \|v_j^* x\|^2$$

length of projection of x onto v_j squared

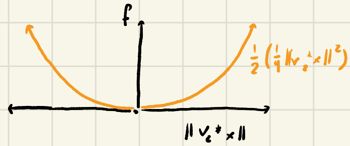
LS: $f(x) = \sum_{j=1}^{n} \frac{1}{2} \sigma_j(A)^2 \underbrace{\|v_j^*(x - x_*)\|^2}_{\substack{\text{length of projection} \\ \text{of residual onto } v_j \text{ square}}}$

• **Ex:** $\lambda_1(M) = \sigma_1(A)^2 = 9$, $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\qquad$ $\lambda_2(M) = \sigma_2(A)^2 = \frac{1}{4}$, $v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

• along v_1:



• along v_2:

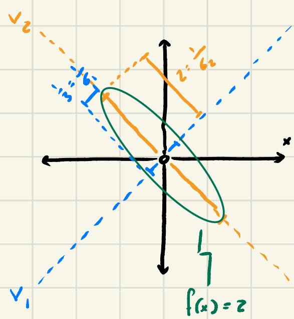


$\lambda_1(M) = \sigma_1(A)^2$ large $\Rightarrow$ f sensitive along v_1 $\qquad$ $\lambda_2(M) = \sigma_2(A)^2$ small $\Rightarrow$ f insensitive along v_2

• curvature along v_j determined by $\lambda_j(M)$ (if $\lambda_j(M) > 0$ curve up, $= 0$ flat, < 0 curve down)
(M p.d. $\Leftrightarrow \lambda_j(M) > 0 \Leftrightarrow f$ convex)

• plot $f(x)$ using isoclines (level sets)

$$f(x) = c \Rightarrow \frac{1}{2} x^* M x = c \Rightarrow \sum_{j=1}^{n} \lambda_j(M) \|v_j^* x\|^2 = \frac{1}{2} c$$



• **Ex:** $9\|v_1^* x\|^2 + \frac{1}{4}\|v_2^* x\|^2 = 2c$ ... equation for an ellipse!

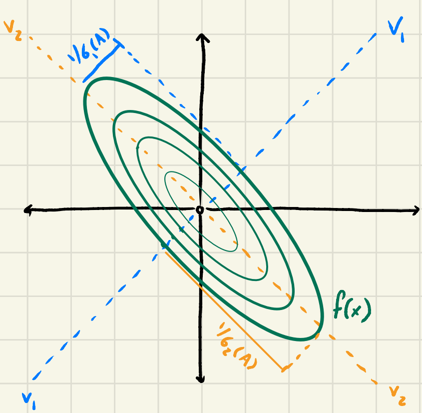
• principal axes $= v_1, v_2, \dots$ of relative length $\sqrt{1/\lambda_j(M)} = 1/\sigma_j(A)$

• try $f(x) = 1/2 \Rightarrow 9\|v_1^* x\|^2 + \frac{1}{4}\|v_2^* x\|^2 = 1$

if:

• $x \parallel v_1 \Rightarrow v_1^* x = \|x\|^2$, $\underset{\substack{\downarrow \\ v_2 \perp v_1}}{v_2^* x = 0}$ so $9\|x\|^2 = 1$ so $\|x\| = 1/3$ $(= 1/\sigma_1(A))$

• $x \parallel v_2 \Rightarrow v_2^* x = \|x\|^2$, $v_1^* x = 0$ so $\frac{1}{4}\|x\|^2 = 1$ so $\|x\| = 2$ $(= 1/\sigma_2(A))$



$f(x)$ is a quadratic bowl centered at x_* w/ elliptical isoclines:

• principal axes are $\parallel$ to eigenvectors of M
$\parallel$ to singular vectors of A

• lengths $\propto \sqrt{\frac{1}{\lambda_j(M)}} = \frac{1}{\sigma_j(A)}$

• **Iterative Solution:** $f(x) = \frac{1}{2} x^* M x$ where $M = A^* A$ (x centered so $x_* = 0$)

• sequence of iterates (guesses at soln):

$$x^{(0)} \longmapsto x^{(1)} \longmapsto x^{(2)} \longmapsto \dots \text{ approaching the minimizer } x_* = 0$$

via recursive update rule: $x^{(k+1)} = x^{(k)} + s_k z^{(k)}$

search direction, descent direction if $f(x^{(k+1)}) < f(x^{(k)})$ for sufficiently small s_k

step length

study convergence rate of $\|x^{(k)}\|$ (error $x^{(k)} - x_*$)

$f(x^{(k)})$ (objective)

$\|\nabla f(x^{(k)})\|$ (optimality, $\nabla f(x^{(k)}) \to 0$)

• **1st order methods:** pick $z^{(k)}$ using $\{x^{(j)}\}_{j=0}^{k}$, $\{f(x^{(j)})\}_{j=0}^{k}$, and $\{\nabla_x f(x^{(j)})\}_{j=0}^{k}$

converge linearly: $\|x^{(k+1)}\| = \mathcal{O}(\|x^{(k)}\|) \leq C\|x^{(k)}\|$

so: $\|x^{(k)}\| \leq C^k \|x_0\|$ (typically tight for worst case x)

depends on method

errors/objective/gradients decay geometrically in C. slow if C close to 1.

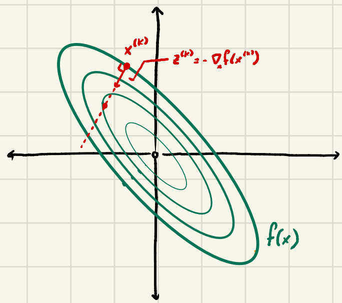
• **2nd order methods:** also use $\{H(x^{(j)})\}_{j=0}^{k}$ ... achieve quadratic convergence: $\|x^{(k+1)} - x_*\| = \mathcal{O}(\|x^{(k)} - x_*\|^2)$

... much faster per step, but: steps are expensive ($\mathcal{O}(n^2)$ to form H, $\mathcal{O}(n^3)$ to use it)

too expensive per step in high dimensions

• most iterative methods for LS are 1st order (objective is already quadratic)

• **Gradient Descent (for LS):** greedy search direction...



• **idea:** pick $z^{(k)}$ to be the direction of fastest descent...

$$z^{(k)} \parallel -\nabla_x f(x)$$

then: $x^{(k+1)} = x^{(k)} - s_k \nabla_x f(x^{(k)})$ ← G.D. generic

• for LS: $\nabla_x f(x) = \nabla_x \frac{1}{2} x^* M x = Mx$

so: $x^{(k+1)} = x^{(k)} - s_k (M x^{(k)}) = (I - s_k M) x^{(k)}$. ← G.D. quadratic

$A^* A$ for LS

• **Question:** how to choose the step size s_k?

- **Question: how to choose the step size s_k?**

  - fix $s_k = s$ ahead of time (convergence rate will depend on "learning rate" s)

  - if $s_k = s$ then $\boxed{x^{(k+1)}} = x^{(k)} - s\,\nabla_x f(x^{(k)}) = \boxed{x^{(k)} - s\,M x^{(k)}} = (I - sM)\,x^{(k)}$

    so:

    $$x^{(k)} = (I - sM)^k x^{(0)}.$$ to optimize convergence, pick s to minimize $\max_j \{ |1 - s\lambda_j(M)| \} = \max_j \{ |1 - s\sigma_j(A)^2| \}$

    optimal s is chosen s.t.

    $$|1 - s\lambda_{min}(M)| = |1 - s\lambda_{max}(M)| \Rightarrow \boxed{s = \frac{2}{\lambda_{max}(M) + \lambda_{min}(M)} = \frac{2}{\sigma_1^2(A) + \sigma_n^2(A)}}$$

    (note: optimizing s requires spectral info... can be expensive to get, may require "direct" analysis...)

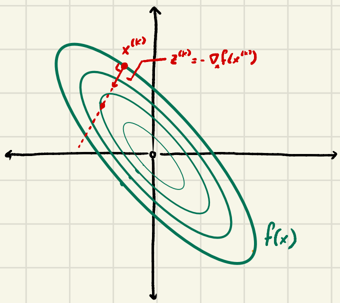
    then: $\|x^{(k+1)} - x_*\| \le C\,\|x^{(k)} - x_*\|$ so $\|x^{(k)} - x_*\| \le C^k\,\|x^{(0)} - x_*\|$

    where:

    $$C = \frac{\lambda_{max}(M) - \lambda_{min}(M)}{\lambda_{max}(M) + \lambda_{min}(M)} = \frac{\sigma_1^2(A) - \sigma_n^2(A)}{\sigma_1^2(A) + \sigma_n^2(A)} = \underbrace{1 - \mathcal{O}(\kappa(M)^{-1}) = 1 - \mathcal{O}(\kappa(A)^{-2})}_{\text{very slow when ill-conditioned}}$$

  - so, for fixed s, the best case method (choice of s) has worst case rate $C = 1 - \mathcal{O}(\kappa(A)^{-2})$.
    (and the bound is tight, achieved for worst-case/adversarial $x^{(0)}$)

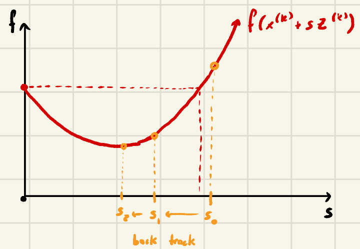
  - choose s_k adaptively at each step.



    - Line Search: if $z^{(k)}$ is a descent direction then $\exists$ s sufficiently small s.t.

      $$f(x^{(k)} + s\,z^{(k)}) < f(x^{(k)})$$

      - picking s is a 1-D optimization problem...
      - solve approximately by back tracking...
        - start w/ large s
        - iteratively reduce s (backtrack) until $f(x^{(k)} + s\,z^{(k)})$ is sufficiently small relative to $f(x^{(k)})$.



    - Exact line search: pick s_k to minimize $f(x^{(k)} + s\,z^{(k)})$

      (greedy step size)

      - aim: $s_k = \arg\min_s \{ f(x^{(k)} + s\,z^{(k)}) \} = \arg\min_s \{ f(x^{(k)} - s\,\nabla_x f(x^{(k)})) \}$

      - for $f(x) = \frac{1}{2} x^* M x$, $\nabla_x f(x) = Mx \Rightarrow s_k = \dfrac{x^{(k)*} M^2 x^{(k)}}{x^{(k)*} M^3 x^{(k)}}$ $\leftarrow \|\underbrace{\nabla_x f(x^{(k)})}_{g^{(k)}}\|^2$; $\leftarrow g^{(k)*} M g^{(k)}$

• GD w/ exact line search:

$$x^{(k+1)} = (I - s_k M)\, x^{(k)} \quad \text{where} \quad \underbrace{s_k = \frac{x^{(k)*} M^2 x^{(k)}}{x^{(k)*} M^3 x^{(k)}}}_{\text{no need for spectral info}}$$

implementation: given $x^{(k)}$

(i) compute $g^{(k)} = M x^{(k)}$

(ii) compute $\alpha = \|g^{(k)}\|^2$, $\beta = g^{(k)*}(M g^{(k)})$

(iii) set $s_k = \alpha/\beta$

(iv) $x^{(k+1)} = x^{(k)} - s_k g^{(k)}$

notice: solve an inverse problem $Ax_* = b$ using only forward multiplication by A and A^*. Cheap if A is sparse.

• using a greedy step can speed convergence but...

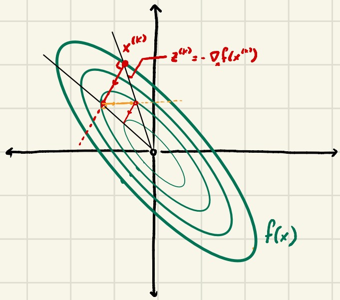
• Convergence is still linear w/ $\|x^{(k+1)} - x_*\| \le C\, \|x^{(k)} - x_*\|$

and

$$C = \frac{\lambda_{max}(M) - \lambda_{min}(M)}{\lambda_{max}(M) + \lambda_{min}(M)} = 1 - \mathcal{O}(\kappa(M)^{-1}) = 1 - \mathcal{O}(\kappa(A)^{-2})$$

is the same as for (best) fixed step-size and is achieved for adversarial/worst-case input.

• in worst-case, no better than fixed step rule...



G.D. follows a "zig-zag" path...

• Worst-case $x^{(0)}$ is $x^{(0)}$ parallel to $\lambda_{min}(M)\, v_1 + \lambda_{max}(M)\, v_2 = \sigma_n(A)^2 v_1 + \sigma_1(A)^2 v_2 \dots$

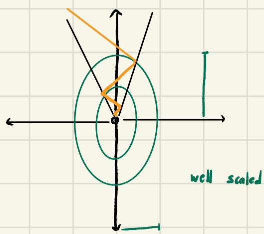
• If A is ill-conditioned $\kappa(A)$ is large $\kappa(M)^{-1} = \kappa(A)^{-2}$ very small, so C is close to one. Converge very slowly.

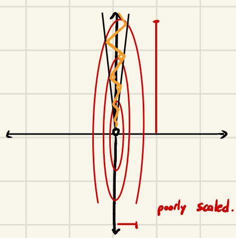
  • the issue is scaling. if A is ill-conditioned $f(x)$ is poorly scaled, far more responsive/sensitive to some variables than others

    • as a result, get large gradients pointing in almost entirely the wrong direction!

    • zig-zag making many over-corrections to the sensitive variables while making extremely slow progress on the insensitive variables!!

• the worse the scaling the slower convergence.





- Momentum & Acceleration: (avoid zig-zagging)

  - Momentum: (Polyak) incorporate past search direction into current search direction

  - Acceleration: (Nesterov) look ahead along past search direction to evaluate gradient

- W/ optimized parameters converge linearly w/ $C = 1 - \mathcal{O}(\kappa_2(M)^{-1/2}) = 1 - \mathcal{O}(\kappa_2(A)^{-1})$

  (vs. G.D: $1 - \mathcal{O}(\kappa_2(M)^{-1}) = 1 - \mathcal{O}(\kappa_2(A)^{-2})$)

  - better, converge at square root of conditioning of M, conditioning of A
    - still slow if very ill-conditioned...

- Question: Can we do better?

Week 7 - Iterative Methods and Eigenvalue Problems

Tuesday - 05/02/2023 - Conjugate Gradient Descent and Iterative LS Solvers

- Logistics:
  - HW 4 posted, due this Thursday
  - HW 5 posted, due next Thursday
- Goals:
  - Improving Gradient Descent:
    - Momentum and acceleration
    - Conjugate Gradient Descent: ** near optimal **
      - Iterative Methods and Matrix Polynomials
      - Minimizing the residual over subspaces
      - Conjugacy & Conjugate Bases
      - Conjugate Gradient Descent

- Gradient Descent is Slow: aim, minimize $f(x) = \frac{1}{2}(x - x_*)^T M (x - x_*)$ for M symmetric, positive definite

  for LS problem: $M = A^*A$, $x_* = (A^*A)^{-1}A^*b$

  - last week we saw that GD converges slowly when A or M are ill-conditioned,

$$\|x^{(k)} - x_*\|_2 \leq C^k \|x^{(0)} - x_*\|_2$$

  where:

$$\left.\begin{aligned} C &= \frac{\lambda_{max}(M) - \lambda_{min}(M)}{\lambda_{max}(M) + \lambda_{min}(M)} = 1 - \mathcal{O}(\kappa(M)^{-1}) \\ &= \frac{\sigma_{max}^2(A) - \sigma_{min}^2(A)}{\sigma_{max}^2(A) + \sigma_{min}^2(A)} = 1 - \mathcal{O}(\kappa(A)^{-2}) \end{aligned}\right\} C \approx 1 \text{ when ill-conditioned}$$

- Momentum & Acceleration: (avoid zig-zagging) try adjusting our search direction...
  - Momentum: incorporate past search direction into current search direction (Polyak)

$$x^{(k+1)} = x^{(k)} + s_k z^{(k)}, \quad z^{(k)} = -\nabla_x f(x^{(k)}) + t_k z^{(k-1)}, \quad t_1 = 0$$

  - Acceleration: look ahead along past search direction to evaluate gradient (Nesterov)

- w/ fixed parameters converge linearly w/ optimal $C = 1 - \mathcal{O}(\kappa(M)^{-1/2}) = 1 - \mathcal{O}(\kappa(A)^{-1})$

  (vs. GD: $1 - \mathcal{O}(\kappa(M)^{-1}) = 1 - \mathcal{O}(\kappa(A)^{-2})$)

  - better, converge at square root of conditioning of M, conditioning of A
    - still slow if very ill-conditioned... Can we do better?

- But, the optimal momentum method w/ adaptive parameters converges in exactly n steps!

• how ??

• given: $x^{(k+1)} = x^{(k)} + s_k z^{(k)}$

$$z^{(k)} = -\nabla_x f(x^{(k)}) + t_k z^{(k-1)}$$

pick t_k s.t. the sequence of search directions

$$z^{(0)} = -\nabla_x f(x^{(0)}),\ z^{(1)},\ z^{(2)}, \dots$$

are all "conjugate" ($\perp_M$)

• Def: given $M \in \mathbb{C}^{n\times n}$, Hermitian, positive definite then the product $\langle u, v\rangle_M = u^T M v$ is an inner product and induces a norm on $\mathbb{C}^n$, $\|u\|_M^2 = \langle u, u\rangle_M = u^T M u$

• Def: we say u and v are conjugate if $u \perp_M v$. That is $\langle u, v\rangle_M = u^T M v = 0$.

• aim to work w/ conjugate search directions ($\perp$ under $\langle \cdot, \cdot \rangle_M$)

• note: can build a set of conjugate directions

$$q_1, q_2, q_3, \dots$$

from a sequence of proposal directions

$$v_1, v_2, v_3, \dots$$

s.t. $\text{span}(q_1, q_2, \dots q_k) = \text{span}(v_1, \dots v_k)$ via Gram-Schmidt using $\langle \cdot, \cdot \rangle_M$

• $q_k \parallel$ to $v_k - \sum_{j=1}^{k-1} \frac{\langle q_j, v_k\rangle_M}{\langle q_j, q_j\rangle_M} q_j$

$\langle q_j, q_j\rangle_M = \|q_j\|_M^2$

• the ensuing algorithm is the momentum method w/ adaptive steps:

$$x^{(k+1)} = x^{(k)} + s_k z^{(k)}$$

$$z^{(k)} = -\nabla_x f(x^{(k)}) + \sum_{j<k} t_{k,j} z^{(j)}$$

that minimizes the objective

$$f(x^{(k)}) = \frac{1}{2}(x^{(k)} - x_*)^T M (x^{(k)} - x_*) = \frac{1}{2}\|A x^{(k)} - b\|^2 + \underset{\text{some constant}}{C}$$

over all possible choices of s, t for all $k \in [1, n]$.

• Moreover, only uses $t_{k,k-1}$ ($t_{k,j} = 0$ if $j < k-1$)

... only adjusts $\nabla_x f(x^{(k)})$ by last search direction

• So, to derive, let's study the sequence of objective function values and choose s, t to minimize the objective at each k...

- let $f(x) = \frac{1}{2}(x - x_*)^T M (x - x_*) = \frac{1}{2}\|Ax - b\|^2 + C$

where $M = A^*A$, $x_* = (A^*A)^{-1}A^*b$

$\nabla_x f(x) = M(x - x_*) = A^*Ax - A^*b$

- let's start w/ just Gradient Descent:

$$x^{(0)}$$
$$\downarrow$$
$$x^{(1)} = x^{(0)} - s_0 \nabla_x f(x^{(0)}) \mapsto x^{(1)} = x^{(0)} - s_0 M(x^{(0)} - x_*)$$
$$\downarrow$$
$$x^{(2)} = x^{(1)} - s_1 \nabla_x f(x^{(1)}) \mapsto x^{(2)} = x^{(1)} - s_1 M(x^{(1)} - x_*)$$

error: $y^{(k)} = x^{(k)} - x_*$, $\nabla_x f(x^{(k)}) = M(x^{(k)} - x_*) = My$

$$y^{(0)} = x^{(0)} - x_*$$
$$\downarrow$$
$$y^{(1)} = x^{(1)} - x_* = (x^{(0)} - x_*) - s_0 \nabla_x f(x^{(0)}) \mapsto x^{(1)} = (x^{(0)} - x_*) - s_0 M(x^{(0)} - x_*)$$
$$y^{(1)} = y^{(0)} - s_0 M y^{(0)} = (I - s_0 M) y^{(0)}$$
$$\downarrow$$
$$y^{(2)} = y^{(1)} - s_1 M y^{(1)} = (I - s_1 M) y^{(1)} = (I - s_1 M)(I - s_0 M) y^{(0)}$$
$$= [I - (s_0 + s_1) M + s_0 s_1 M^2] y^{(0)}$$
$$= [M^0 - (s_0 + s_1) M^1 + s_0 s_1 M^2] y^{(0)}$$

$$y^{(2)} = \underbrace{[M^0 - (s_0 + s_1) M^1 + s_0 s_1 M^2]}_{p_2(M \mid s)} y^{(0)}$$

$$p_2(\xi \mid s) = \xi^0 - (s_0 + s_1)\xi^1 + s_0 s_1 \xi^2$$

$$\downarrow$$

$$y^{(k+1)} = y^{(k)} - s_k M y^{(k)} = (I - s_k M) y^{(k)}$$
$$= p_{k+1}(M \mid s)\, y^{(0)}$$

Conclusion: error at step K, $y^{(k)} = x^{(k)} - x_*$

is a polynomial of degree K in M

w/ coefficents determined by $s_0, s_1, \ldots$

times $y^{(0)} = x^{(0)} - x_*$

- $y^{(k+1)} = p_k(M \mid s)\, y^{(0)}$

$$p_k(\xi \mid s) = \prod_{j=0}^{k} (1 - s_j \xi)$$

} when optimizing over the choice of s

the goal is to make $\|p_k(M \mid s)\|_2$ as small as possible

$$\|y^{(k)}\|_2 \le \|p_k(M \mid s)\|_2 \, \|y^{(0)}\|_2$$

[= is tight

use momentum:

$$y^{(k+1)} = y^{(k)} - s_k z^{(k)}$$

$$z^{(k)} = \underbrace{-\nabla_x f(x^{(k)})}_{My^{(k)}} + \sum_{j<k} t_{kj} z^{(j)}$$

then:

$$y^{(k+1)} = \overbrace{p_k}^{\text{some polynomial degree } k \text{ in } M}(M \mid s, t)\, y^{(0)}$$

$$\left[I + c_1(s,t) M + c_2(s,t) M^2 + \dots c_k(s,t) M^k \right] y^{(0)}$$

always just I not $c_0(s,t) I$ since $z^{(0)} = -\nabla_x f(x^{(0)}) = My^{(0)}$

$$y^{(k+1)} = y^{(0)} + c_1(s,t) My^{(0)} + c_2(s,t) M^2 y^{(0)} + \dots c_k(s,t) M^k y^{(0)}$$

$$= y^{(0)} + \begin{bmatrix} | & | & & | \\ My^{(0)} & M^2 y^{(0)} & \dots & M^k y^{(0)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} c_1(s,t) \\ c_2(s,t) \\ \vdots \\ c_k(s,t) \end{bmatrix}$$

1. we'd like (eventually) $\left[My^{(0)} \; M^2 y^{(0)} \dots M^k y^{(0)} \right] \vec{c}(s,t) = -y^{(0)}$

   for some k

2. $y^{(k+1)} \in y^{(0)} + \underbrace{\text{span}\{ My^{(0)}, M^2 y^{(0)}, \dots M^k y^{(0)} \}}_{\mathcal{K}_k(M, y^{(0)}) = \text{"Krylov Subspace"}}$

• Fact: if M has n distinct eigenvalues, then

for almost any input vector $y^{(0)}$

$\dim(\mathcal{K}_k(M, y^{(0)})) = \min(k, n)$

then since $M \in \mathbb{C}^{n \times n}$, $M^j y^{(0)} \in \mathbb{C}^n$, so $\mathcal{K}_k(M, y^{(0)}) \in \mathbb{C}^n$

so, if $k \geq n$

$$\left.\begin{aligned} &\mathcal{K}_k(M, y^{(0)}) \subseteq \mathbb{C}^n \\ &\dim(\mathcal{K}_k(M, y^{(0)})) = n \end{aligned}\right\} \Rightarrow \text{for } k \geq n,\ \mathcal{K}_k(M, y^{(0)}) = \mathbb{C}^n.$$

• so, ... after n steps $\mathcal{K}_k(M, y^{(0)}) = \mathbb{C}^n$

thus contains any vector in $\mathbb{C}^n$

in particular, it contains $-y^{(0)}$...

then $\exists$ a coefficient vector $\vec{c}$ s.t.

$$y^{(n)} = y^{(0)} + \underbrace{\left[My^{(0)} \; M^2 y^{(0)} \dots M^n y^{(0)} \right] \vec{c}}_{= -y^{(0)}} = y^{(0)} - y^{(0)} = 0.$$

i.e. $x^{(n)} = x_*$, solve in n steps.

• thus, we hope to choose s, t s.t. after n steps the error $y^{(n)} = 0$.

- **Moral:** $y^{(k)} \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})$ ← space of possible errors left over after k steps of a momentum method...

for $k \geq n$ it's possible to find $-y^{(0)} \in \mathcal{K}_k(M, y^{(0)})$
thus to solve the optimization problem exactly.

- **Optimality for each stage k:** momentum method w/ s, t is optimal if

$$y^{(k+1)} = \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ f(y) \} = \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ \tfrac{1}{2} y^T M y \}$$

- given a conjugate basis for $\mathcal{K}_k(M, y^{(0)})$, $Q^{(k)} = \{q^{(1)}, q^{(2)}, \dots q^{(k)}\}$
it is possible to solve for $y^{(k+1)}$ exactly...
  - Suppose $Q^{(k)} = \{q^{(1)}, \dots q^{(k)}\}$ s.t. $\operatorname{span}(Q^{(k)}) = \mathcal{K}_k(M, y^{(0)})$
  and $q^{(i)} \perp_M q^{(j)}$ if $i \neq j$

$$\left.\begin{aligned} y^{(k+1)} &\in y^{(0)} + \mathcal{K}_k(M, y^{(0)}) = y^{(0)} + \sum_{j=1}^{k} \alpha_j q^{(j)} \\ y^{(0)} &\in \mathbb{C}^n = \mathcal{K}_n(M, y^{(0)}) = \sum_{j=1}^{n} \beta_j q^{(j)} \end{aligned}\right\} y^{(k+1)} = \sum_{j=1}^{k} (\beta_j + \alpha_j) q^{(j)} + \sum_{j=k+1}^{n} \beta_j q^{(j)}$$

if the goal is to make $y^{(k+1)}$ as close to 0 as possible then, choose

$\alpha_j = -\beta_j \dots$

- **Suggests an outline for a method:**

run GD + momentum w/ adaptive step sizes:

$$x^{(k+1)} = x^{(k)} + s_k z^{(k)}, \quad z^{(k)} = -\nabla_x f(x^{(k)}) + \sum_{j<k} t_{kj} z^{(j)}$$

where s and t are chosen s.t. the errors satisfy

$$y^{(0)} = x^{(0)} - x_*,\ y^{(k)} = x^{(k)} - x_* = \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ f(y) \} = y^{(0)} - \sum_{j=1}^{k} \beta_j q^{(j)} = y^{(k-1)} - \underbrace{\beta_k q^{(k)}}_{-s_k z^{(k)},\ z^{(k)} = q^{(k)}}$$

↑ best possible for all iterative methods using momentum

where:
- $Q^{(k)} = \{q^{(1)}, q^{(2)}, \dots q^{(k)}\}$ is a $\perp_M$ basis
to $\mathcal{K}_k(M, y^{(0)}) = \{ M y^{(0)}, M^2 y^{(0)}, \dots M^k y^{(0)} \} = \operatorname{span}\{ \nabla_x f(y^{(0)}), \nabla_x f(y^{(1)}), \dots \nabla_x f(y^{(k-1)}) \}$
- β satisfies $y^{(0)} = \sum_{j=1}^{n} \beta_j q^{(j)}$ (coefficients of expansion of initial error onto the conjugate basis)

Thursday - 05/04/2023 - Conjugate Gradients and Power Iteration

- **Logistics:**
  - Reading posted
  - HW 4 due tonight
  - HW 5 due next week

- **Goals:**
  - Conjugate gradient descent

- **Last Class:** given $f(x) = \frac{1}{2}(x - x_*)^* M (x - x_*) = \frac{1}{2}\|Ax - b\|^2$, $A \in \mathbb{C}^{m \times n}$ full rank
  M hermitian positive definite

  where $M = A^*A$, $\underset{\uparrow \text{ unknown}}{x_*} = (A^*A)^{-1}A^*b$

  then an iterative momentum-based method sets:

$$x^{(k+1)} = x^{(k)} + s_k z^{(k)} \quad \text{where} \quad z^{(k)} = \underbrace{-\nabla_x f(x^{(k)})}_{-M(x - x_*) = -(A^*Ax - A^*b)} + \sum_{j<k} t_{kj} z^{(j)}$$

(includes all GD methods by setting $t_{kj} = 0 \; \forall \; j<k$)

has errors: $y^{(k)} = x^{(k)} - x_*$, $\left(f(y) = \frac{1}{2} y^* M y, \; \nabla_y f(y) = My \right)$

contained in the sequence of affine subspaces:

$$\boxed{y^{(k)} \in \underset{\uparrow \text{ initial error}}{y^{(0)}} + \underbrace{K_k(M, y^{(0)})}_{\text{the space we are moving in}}}$$

i.e. $y^{(k)} = y^{(0)} + \sum_{j=1}^{k} (M^j y^{(0)}) \, c_j(s,t)$

where the coefficients $c(s,t)$ are determined by the recursive rule for s and t.

where

$$K_k(M, y^{(0)}) = \text{span}\{ My^{(0)}, M^2 y^{(0)}, \dots M^k y^{(0)} \}$$

is the k^{th} Krylov subspace.

- **Fact:** if M has n distinct eigenvalues, then, for almost any $y^{(0)}$
  $K_n(M, y^{(0)}) = \mathbb{C}^n$ so $y^{(0)} \in K_n(M, y^{(0)})$ and, for all $y^{(0)}$, $y^{(0)} \in K_k(M, y^{(0)})$ for some $k \le n$.

- then, any momentum method satisfies:

$$f(y^{(k)}) \geq \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ f(y) \}$$

and the best possible momentum method would set:

$$y^{(k)} = \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ f(y) \}$$ thus would achieve $y^{(k)} = 0 \Rightarrow x^{(k)} = x_*$ for some $K \leq n$ (provided M has simple eigenvalues).

- So, our target is to set/solve:

$$y^{(k)} = \underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} \{ f(y) \}$$ in an iterative fashion (must achieve implicitly via careful choice of s and t)

- write: $y^{(k)} = y^{(0)} + \begin{bmatrix} | & | & & | \\ My^{(0)}, & M^2 y^{(0)}, & \dots & M^K y^{(0)} \\ | & | & & | \end{bmatrix} \vec{c}$

and try to solve for the coefficients $\vec{c}$...

- Problem: the basis $B^{(k)} = \{ M^j y^{(0)} \}_{j=1}^{K}$ is extremely ill-conditioned...

(why? if $M = V \Lambda V^*$, where $A = U \Sigma V^*$, $\Lambda = \Sigma^* \Sigma$ then $V^* = V^{-1}$ so:

$$M^j = (V \Lambda \underbrace{V^*)(V}_{I} \Lambda \underbrace{V^*)(V}_{I} \Lambda \underbrace{V^*) V}_{I} \dots \underbrace{V^* (V}_{I} \Lambda V^*)$$
$$= V \Lambda^j V^* = V \operatorname{diag}(\lambda_1^j, \lambda_2^j, \dots \lambda_n^j) V^* = \sum_{i=1}^{n} \lambda_i^j [v_i v_i^*]$$

so $$M^j y^{(0)} = V \Lambda^j V^* y = \sum_{i=1}^{n} (\lambda_i^j (v_i^* y^{(0)})) \begin{matrix} | \\ v_i \\ | \end{matrix}$$

magnitude will either $\nearrow$ to ∞ if $\lambda_i > 1$, or $\searrow$ 0 if $\lambda_i < 1$...

$|\lambda_1| > |\lambda_i| \ \forall \ i > 1$ so:

$$\frac{1}{\lambda_1^j} M^j y^{(0)} = (v_1^* y^{(0)}) \begin{matrix} | \\ v_1 \\ | \end{matrix} + \sum_{i>1} \underbrace{\left(\frac{\lambda_i}{\lambda_1}\right)}_{<1}^j (v_i^* y^{(0)}) \begin{matrix} | \\ v_i \\ | \end{matrix}$$

so: $\frac{1}{\lambda_1^j} M^j y^{(0)}$ converges to $(v_1^* y^{(0)}) v_1$ at $\mathcal{O}\left(\left(\frac{\lambda_2}{\lambda_1}\right)^j\right)$

thus: as $j \to$ large, the columns of $M^j y^{(0)}$ become $\sim$ parallel)

- so, need a better basis for $\mathcal{K}_K(M, y^{(0)})$...

- <u>idea</u>: leverage orthogonality... really, conjugacy (orthogonality w.r.t. M)

  - <u>recall</u>: - <u>Def</u>: given $M \in \mathbb{C}^{n\times n}$, Hermitian, positive definite then the product $\langle u, v\rangle_M = u^T M v$ is an inner product and induces a norm on $\mathbb{C}^n$, $\|u\|_M^2 = \langle u, u\rangle_M = u^T M u$

  - <u>Def</u>: we say u and v are <u>conjugate</u> if $u \perp_M v$. That is $\langle u, v\rangle_M = u^T M v = 0$.

- Let $Q^{(k)} = [q^{(1)}, q^{(2)}, \dots q^{(k)}]$ be an $\perp_M$ basis for $\mathcal{K}_K(M, y^{(0)})$

  (recall, can build by applying G.S. to any sequence of proposal directions: $v^{(1)}, v^{(2)}, \dots$ s.t.

  $$\text{span}(\{v^{(j)}\}_{j=1}^K) = \mathcal{K}_K(M, y^{(0)}) = \text{span}(\{M^j y^{(0)}\}_{j=1}^K)\dots \longleftarrow$$

  <u>Ex</u>: let $v^{(j)} = M^j y^{(0)}$) or $v^{(j)} = \frac{1}{\|M^j y^{(0)}\|} M^j y^{(0}$

  then:

  $$q^{(j)} \text{ is } \parallel \text{ to } v^{(j)}_{\perp_M \mathcal{K}_{j-1}} = v^{(j)}_{\perp_M Q^{(j-1)}} = v^{(j)} - \sum_{i<j} \frac{\langle q^{(i)}, v^{(j)}\rangle_M}{\langle q^{(i)}, q^{(i)}\rangle_M} q^{(i)}$$

- <u>Then</u>: 1. <u>Conjugacy</u>: $q^{(i)} \perp_M q^{(j)}\ \forall i \neq j$ so $\langle q^{(i)}, q^{(j)}\rangle_M = q^{(i)*} M q^{(j)} = 0$

  2. <u>Change of Basis</u>: if $w \in \mathcal{K}_K(M, y^{(0)})$ for some K then $\exists$ coefficients $\hat{w} = [\hat{w}_1, \hat{w}_2, \dots \hat{w}_K]$ s.t.

  $$w = Q^{(K)}\hat{w} = \sum_{j=1}^K \hat{w}_j q^{(j)}$$

  and, $\forall\ w \in \mathcal{K}_K(M, y^{(0)})$, $w = Q^{(K)}\hat{w}$

  we can recover $\hat{w}$ from w via inner-products:

  $$\hat{w}_j \propto \langle q^{(j)}, w\rangle_M = \langle q^{(j)}, Q^{(K)}\hat{w}\rangle_M = \langle q^{(j)}, \sum_{i=1}^K \hat{w}_i q^{(i)}\rangle_M$$
  $$= \sum_{i=1}^K \hat{w}_i \underbrace{\langle q^{(j)}, q^{(i)}\rangle_M}_{=0 \text{ unless } i=j} = \hat{w}_j \langle q^{(j)}, q^{(j)}\rangle_M = \|q^{(j)}\|_M^2\, \hat{w}_j$$

  so: $$\hat{w}_j = \frac{1}{\|q^{(j)}\|_M^2}\langle q^{(j)}, w\rangle_M = \frac{q^{(j)*} M w}{q^{(j)*} M q^{(j)}}$$

  - then: $y^{(K)} = y^{(0)} + \sum_{j=1}^K \hat{y}_j^{(K)}(s,t)\, q^{(j)}$ for some $\hat{y}_j$ determined by s and t

  and: $y^{(0)} = \sum_{j=1}^n \hat{y}_j^{(0)} q^{(j)}$ where $\hat{y}_j^{(0)} = \frac{1}{\|q^{(j)}\|_M^2}\langle q^{(j)}, y^{(0)}\rangle_M$

3. **Objective:** $f(w) = \frac{1}{2} w^* M w$

if $w = Q^{(k)} \hat{w}$ then:

$$f(w) = \frac{1}{2} (Q^{(k)} \hat{w})^* M (Q^{(k)} \hat{w})$$

$$= \frac{1}{2} \left(\sum_{j=1}^{k} \hat{w}_j q^{(j)} \right)^* M \left(\sum_{i=1}^{k} \hat{w}_i q^{(i)} \right) = \frac{1}{2} \sum_{i,j=1}^{k} \bar{\hat{w}}_j \hat{w}_i \left(q^{(j)*} M q^{(i)} \right)$$

$$= \frac{1}{2} \sum_{i,j=1}^{k} \bar{\hat{w}}_j \hat{w}_i \underbrace{\langle q^{(j)}, q^{(i)} \rangle_M}_{=0 \text{ unless } i=j} = \frac{1}{2} \sum_{j=1}^{k} \| q^{(j)} \|_M^2 \, |\hat{w}_j|^2$$

so, if we normalize the basis...

- set $q^{(j)}$ s.t. $\| q^{(j)} \|^2 = 1$

then:

$$f(w) = \frac{1}{2} \sum_{j=1}^{k} |\hat{w}_j|^2 = \frac{1}{2} \| \hat{w} \|^2$$

- so: $f(y^{(k)}) = f\left(y^{(0)} + \sum_{j=1}^{k} \hat{y}_j^{(k)} q^{(j)}\right)$

$$= f\left(\sum_{j=1}^{\hat{n}} \hat{y}_j^{(0)} q^{(j)} + \dots \right) = f\left(\sum_{j=1}^{k} (\hat{y}_j^{(0)} + \hat{y}_j^{(k)}) q^{(j)} + \sum_{j=k+1}^{\hat{n}} \hat{y}_j^{(0)} q^{(j)} \right)$$

$$= \frac{1}{2} \left(\sum_{j=1}^{k} | \underbrace{\hat{y}_j^{(0)}}_{\text{fixed by } y^{(0)}} + \underbrace{\hat{y}_j^{(k)}}_{\text{control w/ s and t}} |^2 + \sum_{j>k} | \underbrace{\hat{y}_j^{(0)}}_{\text{fixed by } y^{(0)}} |^2 \right)$$

- so: $\underset{y \in y^{(0)} + \mathcal{K}_k(M, y^{(0)})}{\operatorname{argmin}} (f(y))$ is achieved by setting:

$$\hat{y}_j = -\hat{y}_j^{(0)} \quad \text{for all } j \le k$$

- aim to choose s, t such that:

$$\hat{y}_j^{(k)}(s,t) = -\hat{y}_j^{(0)}$$

then:

$$\underbrace{y^{(k)} = \underset{y \in y^{(0)} + \mathcal{K}_k}{\operatorname{argmin}} (f(y))}_{\text{optimal over all momentum methods}} \text{ and } f(y^{(k)}) = \sum_{j>k} | \overbrace{\hat{y}_j^{(0)}}^{= \langle q^{(j)}, y^{(0)} \rangle_M} |^2 \text{ and } \underbrace{f(y^{(k)}) = 0} \text{ for some } k \le n$$

so $x^{(k)} = x_*$ for some $k \le n$...

solve in finitely many steps.

4. **Iteration:**
$$y^{(k)} = y^{(0)} - \sum_{j=1}^{k} \hat{y}_j^{(0)} q^{(j)} = \sum_{j=1}^{\hat{n}} \hat{y}_j^{(0)} q^{(j)} - \sum_{j=1}^{k} \hat{y}_j^{(0)} q^{(j)} = \sum_{j>k} \hat{y}_j^{(0)} q^{(j)}$$

$$= y^{(k-1)} - \hat{y}_k^{(0)} q^{(k)} = y^{(k-1)} - \underbrace{\langle q^{(k)}, y^{(0)} \rangle_M}_{= \langle q^{(k)}, y^{(k-1)} \rangle_M \text{ since } y^{(k-1)} = \sum_{j=k}^{\hat{n}} \hat{y}_j^{(0)} q^{(j)}} q^{(k)} = y^{(k-1)} - \langle q^{(k)}, y^{(k-1)} \rangle_M q^{(k)}$$

so:
$$y^{(k)} = \begin{cases} \sum_{j>k} \hat{y}_j^{(0)} q^{(j)} & \leftarrow \text{theory} \\ \text{and} & \\ y^{(k-1)} - \langle q^{(k)}, y^{(k-1)} \rangle_M q^{(k)} & \leftarrow \text{practical iteration: } (x^{(k)} - x_*) = (x^{(k-1)} - x_*) - \langle q^{(k)}, x^{(k-1)} - x_* \rangle_M q^{(k)} \end{cases}$$

so: $x^{(k)} = x^{(k-1)} - \left(q^{(k)*} M (x^{(k-1)} - x_*) \right) q^{(k)}$

$$= x^{(k-1)} - \left(q^{(k)*} \underbrace{(A^* A x^{(k-1)} - A^* b)}_{\nabla_x f(x^{(k-1)})} \right) q^{(k)}$$

- So, practical iteration rule:

given $A, b \Rightarrow M = A^*A$,

$\nabla_x f(x) = Mx - A^*b \leftarrow$ "residual"

given $x^{(k)}$ iterate: $\nabla_x f(x^{(k)}) = Mx^{(k)} - A^*b = r^{(k)}$

$$x^{(k+1)} = x^{(k)} + s_k z^{(k)}$$

where:

1. search direction: $z^{(k)} = -q^{(k)}$ $\leftarrow$ k^{th} conjugate basis vector of $\mathcal{K}_k(M, y^{(0)})$
2. step size: $s_k = \langle q^{(k)}, \nabla_x f(x^{(k)})\rangle = q^{(k)*}\underbrace{(Mx^{(k)} - A^*b)}_{\text{residual } r^{(k)}} = q^{(k)*} r^{(k)}$

- if q's are not normalized,

$$s_k = (q^{(k)*} r^{(k)}) / (q^{(k)*} M q^{(k)})$$

notice: $x^{(k)} = x^{(k-1)} - s_{k-1} q^{(k-1)}$

$$Mx^{(k)} - y = (Mx^{(k-1)} - y) - s_{k-1} M q^{(k-1)}$$
$$= r^{(k-1)} - s_{k-1} M q^{(k-1)}$$

- Substituting in for the residual:

at stage $k \Rightarrow$ have $x^{(k)}, r^{(k-1)}, s_{k-1}$

- introduce $z^{(k)} = q^{(k)}$ $\leftarrow$ don't know how yet

(i) compute $Mq^{(k)}$

(ii) compute $r^{(k)} = r^{(k-1)} - s_{k-1}(Mq^{(k)}) = \nabla_x f(x^{(k)})$

(iii) compute:

$$s_k = \langle q^{(k)}, r^{(k)}\rangle / \|q^{(k)}\|_M^2 = \frac{q^{(k)*} r^{(k)}}{q^{(k)*}(Mq^{(k)})}$$

(iv) update:

$$x^{(k+1)} = x^{(k)} - s_k q^{(k)}$$

- almost a full algorithm...

haven't specified how to compute the conjugate search directions $q^{(k)}$...

- recall: given $Q^{(k-1)} = \{q^{(j)}\}_{j=1}^{k-1}$ we only need a proposal direction $v^{(k)}$ with a nonzero projection onto $\mathcal{K}_k / \mathcal{K}_{k-1}$ i.e. onto $M^k y^{(0)}$

- from GD/momentum: propose $v^{(k)} = \nabla_x f(x^{(k-1)}) = r^{(k-1)} = My^{(k-1)}$

we've already shown that: $y^{(k-1)} = \underbrace{p_{k-1}(M|s,t)\, y^{(0)}}_{\text{some polynomial order } k-1 \text{ in } M} + y^{(0)}$

so:

$$My^{(k)} = My^{(0)} + Mp_{k-1}(M|s,t)\, y^{(0)} = \text{some polynomial order } k \text{ in } M$$

so: $v^{(k)} = \nabla_x f(x^{(k-1)}) = r^{(k-1)} \in \mathcal{K}_k(M, y^{(0)}) / \mathcal{K}_{k-1}(M, y^{(0)})$

• then, orthogonalize:

$$\underbrace{q^{(k)}}_{z^{(k)}} = \overbrace{v^{(k)}}^{\nabla_x f(x^{(k)})} - \sum_{j<k} \frac{\langle q^{(j)}, v^{(k)}\rangle_M}{\underbrace{\langle q^{(j)}, q^{(j)}\rangle_M}_{t_{kj}}} \overbrace{q^{(j)}}^{z^{(j)}}$$

but $v^{(k)} = \nabla_x f(x^{(k-1)}) = r^{(k-1)} = My^{(k-1)}$

and $y^{(k-1)} = \sum_{i=k}^{\hat{n}} \hat{y}_i^{(0)} q^{(i)}$

so $\langle q^{(j)}, v^{(k)}\rangle_M = \langle q^{(j)}, My^{(k-1)}\rangle_M = \langle \underset{\substack{\uparrow \\ j<k,\ j\le k-1}}{Mq^{(j)}}, \sum_{i=k}^{\hat{n}} \hat{y}_i^{(0)} q^{(i)}\rangle_M$

so $Mq^{(j)} \in \mathcal{K}_{j+1}$ $\perp_M$ to $\sum_{i=k}^{\hat{n}} \hat{y}_i^{(0)} q^{(i)}$

unless $j = k-1$...

so $\langle q^{(j)}, v^{(k)}\rangle_M = 0$ unless $j = k-1$!

$t_{kj} = 0$ unless $j = k-1$!

so:

$$q^{(k)} = \overbrace{v^{(k)}}^{\nabla_x f(x^{(k)})} - \sum_{j<k} \frac{\langle q^{(j)}, v^{(k)}\rangle_M}{\underbrace{\langle q^{(j)}, q^{(j)}\rangle_M}_{t_{kj}}} \overbrace{q^{(j)}}^{z^{(j)}} = v^{(k)} - \frac{\langle q^{(k-1)}, v^{(k)}\rangle_M}{\langle q^{(k-1)}, q^{(k-1)}\rangle_M} q^{(k-1)}$$

$$= \nabla_x f(x^{(k)}) - t_{k,k-1} z^{(k-1)}$$ ← 1-step momentum rule!

• Conjugate Gradient Descent:

- input $A, b, x^{(0)}$
- compute $Ax^{(0)} - b \Rightarrow r^{(0)} = A^*(Ax - b)$
- compute $M = A^*A$
- let $q^{(0)} = 0$, $s_0 = 0$
- iterate over $k \Rightarrow$ have $x^{(k)}, r^{(k-1)}, s_{k-1}$

  (i) compute $\langle q^{(k-1)}, r^{(k-1)}\rangle_M / \|q^{(k-1)}\|_M^2$ } orthogonalize

  (ii) let $q^{(k)} = r^{(k-1)} - \langle\ldots\rangle/_{\|\ldots\|}\, q^{(k-1)}$

  (iii) compute $Mq^{(k)}$

  (iv) compute $r^{(k)} = r^{(k-1)} - s_{k-1}(Mq^{(k)}) = \nabla_x f(x^{(k)})$

  (v) compute:

  $$s_k = \langle q^{(k)}, r^{(k)}\rangle / \|q^{(k)}\|_M^2 = \frac{q^{(k)*} r^{(k)}}{q^{(k)*}(Mq^{(k)})}$$ } step

  (vi) update:

  $$x^{(k+1)} = x^{(k)} - s_k q^{(k)}$$

Week 8 - Spectral (Eigenvalue) Problems

Tuesday - May 9^{th}

- **Logistics:**
  - HW 5 due Thursday
  - Project 2 post this week
  - Reading posted

- **Goals:**
  - Eigenvalue and SVD problems:
    - Characteristic polynomial ← avoid, extremely unstable
    - Complexity
  - Methods:
    - Tools: Matrix Powers & Rayleigh Quotients
    - Power Iteration
    - Inverse Iteration
    - Rayleigh Quotient iteration

- **Spectral Problems:**

  1. suppos $A \in \mathbb{C}^{n \times n}$ then an eigenpair of A, (v, λ) s.t.

  $$Av = \lambda v$$

  and, if A admits n linearly ind. eigenvector $V = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{bmatrix}$ then:

  $$AV = V\Lambda, \quad \Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots \lambda_n)$$
  $$A = V \Lambda V^{-1}.$$

  - eigenvalue problem ⇒ extract some subset of the eigenpairs (λ_j, v_j)
  - ... decomposition ⇒ recover all (λ_j, v_j) for $j = 1, \ldots n$.

  2. Singular value problems: given any $A \in \mathbb{C}^{m \times n}$ $\exists$ a SVD of A
     - $A = U \Sigma V^*$ where U, V are unitary and Σ is diagonal, real, nonnegative

  - recover the singular values/vectors: (σ_j, u_j, v_j)

- Converting SVD to eigenvalue...
  - given $A \in \mathbb{C}^{m \times n}$...

  - <u>idea</u>: compute: $M = A^* A$, given $A = U \Sigma V^*$

$$M = V \Sigma^* \underbrace{U^* U}_{I} \Sigma V^* = V(\Sigma^* \Sigma) \underset{\substack{\uparrow \\ = V^{-1}}}{V^*} = V(\Sigma^* \Sigma) V^{-1}$$

so eigenvectors of M: $v_j(M) = v_j(A)$

eigenvalues of M: $\lambda_j(M) = \sigma_j^2(A)$

or, compute: $M = A A^*$, then...

eigenvectors of M: $v_j(M) = u_j(A)$

eigenvalues of M: $\lambda_j(M) = \sigma_j^2(A)$

usually avoid... squares the conditioning

- <u>idea</u>: build $H = \begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix}$, notice H is hermitian, $(m+n) \times (m+n)$

then, the eigenvalues & vectors of $H \iff$ the SVD of A

$$H \left[\begin{array}{cc} V & V \\ \hline U & -U \end{array}\right] = \left[\begin{array}{cc} V & V \\ \hline U & -U \end{array}\right] \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

the eigenvectors of H, col's of $\uparrow$
have as blocks the singular vectors of A
and the eigenvalues of H are (in magnitude) the
singular values of A.

- this is stable...
  so, if we can find eigenvalue decomp $\Rightarrow$ perform an SVD.

- <u>Problem</u>: given $A \in \mathbb{C}^{m \times n}$ how to find eigenpairs (v_j, λ_j)?

- Finding eigenpairs:

  - standard story: $Av = \lambda v \Rightarrow Av - \lambda v = 0 \Rightarrow (A - \lambda I)\underset{\uparrow}{v} = 0$ $v \neq 0$

    this means $null(A - \lambda I)$ is nonempty

    so

    $(A - \lambda I)$ is noninvertible $\iff$ $\underbrace{\det(A - \lambda I)}_{\text{polynomial!}} = 0$

    - the eigenvalues λ_j are the roots of the characteristic polynomial:

    $$p(\lambda) = \det(A - \lambda I)$$

    - then $v_j \in null(A - \lambda_j I)$

- there is no analytic formula for the roots if $n > 5$

  if $n > 5$ then there cannot be a direct method...

  - maybe find the roots numerically...

    but, if A is $n \times n$ then $p(\lambda)$ degree n

    roots of an n^{th} degree polynomial are extremely sensitive to its coefficients

    - root finding for large n is extremely ill-conditioned

- Tools:

1. Powers of Matrices:

- a matrix power, A^K for $K \in \mathbb{Z} \leftarrow$ integer is A times itself K times
  - Ex: $A^1, A^2 = AA, A^3 = AAA, \dots$ ← arise naturally in dynamical systems & numerical methods.

if A is diagonalizable let's try converting into the eigenbasis, think about $A = V \Lambda V^{-1}$

then: 1. $A^2 = AA = (V \Lambda V^{-1})(V \Lambda V^{-1})$

$$= V \Lambda \underbrace{V^{-1} V}_{I} \Lambda V^{-1}$$

$$= V \Lambda \Lambda V^{-1} = V \Lambda^2 V^{-1}$$

$\uparrow$ $\mathrm{diag}(\lambda_1^2, \lambda_2^2, \dots \lambda_n^2)$

$A^2 = V \Lambda^2 V^{-1}$

↑ eigenvalues of A^2, (eigenvalues of $A)^2$

2. $A^3 = A A^2 = (V \Lambda V^{-1})(V \Lambda^2 V^{-1})$

$$= V \Lambda \underbrace{V^{-1} V}_{I} \Lambda^2 V^{-1} = V \Lambda^3 V^{-1}$$

$\uparrow$ $\mathrm{diag}(\lambda_1^3, \lambda_2^3, \dots \lambda_n^3)$

$A^3 = V \Lambda^3 V^{-1}$

↑ eigenvalues of A^3, (eigenvalues of $A)^3$

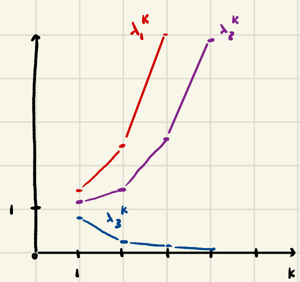
K. $A^K = V \Lambda^K V^{-1}$

↑ $\mathrm{diag}(\lambda_1^K, \lambda_2^K, \dots \lambda_n^K)$

consider: $A^K x = V \Lambda^K \underbrace{V^{-1} x}_{y} = V \Lambda^K y = \underbrace{\sum_{j=1}^{n} (\lambda_j^K y_j) v_j}$

y = coordinates of x in the eigenbasis

dominated by the largest eigenvalue (in magnitude) as $K \to \infty$



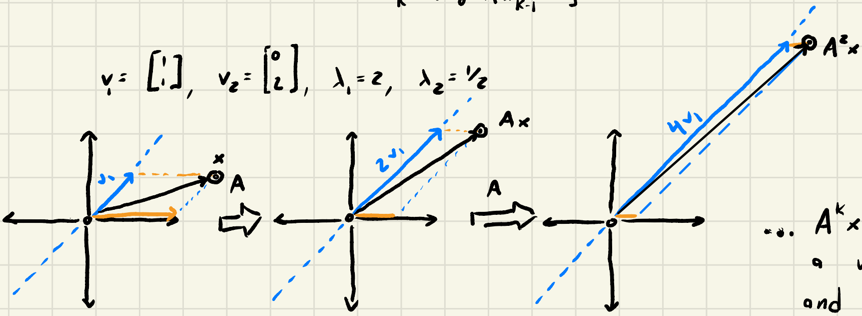
this leads to the power method for computing eigenvalues and eigenvectors

- <u>the power method</u>: idea, start w/ an initial vector x_0

  then compute

$$\left.\begin{aligned} x_1 &= A x_0 \\ x_2 &= A^2 x_0 = A x_1 \\ x_3 &= A^3 x_0 = A x_2 \\ &\vdots \\ x_K &= A^K x_0 = A x_{K-1} \end{aligned}\right\} \text{recursively compute powers}$$

$v_1 = \begin{bmatrix}1\\1\end{bmatrix}$, $v_2 = \begin{bmatrix}0\\2\end{bmatrix}$, $\lambda_1 = 2$, $\lambda_2 = 1/2$



$\therefore$ $A^k x$ converges to a vector parallel to v_1 and grows $\propto \lambda_1^k$

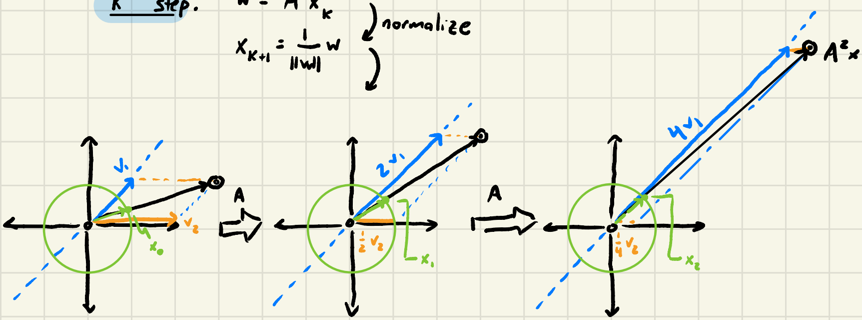
to avoid x_K diverging to ∞ as $K \to \infty$ (largest $\lambda > 1$)

or converging to 0 (largest $\lambda < 1$)

normalize at each iteration:

<u>K^{th} step</u>:

$$\begin{aligned} w &= A x_K && \text{apply } A \\ x_{K+1} &= \frac{1}{\|w\|} w && \text{normalize} \end{aligned}$$



- this idea leads to a practical numerical algorithm for computing eigenvalues and eigenvectors, works for large n, does <u>not</u> require finding roots of a high degree polynomial, much more stable than characteristic polynomial...

  - why? b/c the algorithm is inspired by the actual utility of the decomposition.

2. Rayleigh Quotients:

- how can we estimate eigenvalues from estimated eigenvector?

- Suppose $v \in \mathbb{C}^n$ is $\sim$ an eigenvector

then, want to find a scalar α s.t.

$$Av \approx \alpha v$$

optimize α:

$$\alpha_* = \text{argmin}\{\|Av - \alpha v\|^2\} = \text{argmin}\{\underbrace{v^*A^*Av - 2\alpha v^*Av + \alpha^2 v^*v}_{f(\alpha)}\}$$

to minimize: $\partial_\alpha f(\alpha) = -2v^*Av + 2\alpha v^*v = 0$

requires : $\alpha_* = \dfrac{v^*Av}{v^*v}$

- **Def:** the Rayleigh Quotient $r(A,v) = \dfrac{v^*Av}{v^*v}$

(is the nearest approximation to an eigenvalue of A given approximate eigenvector v)

- Facts: 1. if v is an eigenvector of A w/ eigenvalue λ then:

$$r(A,v) = \frac{v^*Av}{v^*v} = \frac{v^*\lambda v}{v^*v} = \lambda\frac{\|v\|^2}{\|v\|^2} = \lambda.$$

Rayleigh quotient of an eigenvector returns an eigenvalue!

2. if v is an eigenvector with eigenvalue λ

then, given $v' = v + \delta v$: $\delta v = v' - v$

$$r(A,v') = \lambda + \mathcal{O}(\|\delta v\|^2) \longleftarrow r(A,v') - r(A,v) = \mathcal{O}(\|v'-v\|^2)$$

as $\|\delta v\| \to 0$. quadratically accurate!

(why? because $\nabla_x r(A,x)\big|_{x=v} = \dfrac{2}{x^*x}(Ax - r(A,x)x)\big|_{x=v} = \dfrac{2}{v^*v}(Av - \lambda v) = 0$

if $Av = \lambda v$.

- put these ideas together to derive iterative methods for approximating eigenvalues and eigenvectors...

• **Iterative Methods:** Assume $A \in \mathbb{C}^{n \times n}$, hermitian (hence diagonalizable) with simple (non-repeated) eigenvalues

1. **The Power Method:**

   1. input A, $v^{(0)}$ w/ $\|v^{(0)}\|_2 = 1$
   2. iterate until stopping
      - (i) $w = Av^{(k-1)}$ • apply A $(A^{k-1} \to A^k)$
      - (ii) $v^{(k)} = w/\|w\|$ • normalize $\to$ estimate eigenvector
      - (iii) $\lambda^{(k)} = r(A, v^{(k)}) = v^{(k)*}\underbrace{(Av^{(k)})}_{\text{in efficient implementation, store for next step}}$ • estimate eigenvalue

   unlike the characteristic polynomial find $\approx$ eigenvectors first uses natural properties of eigenvalues/vectors

• **Convergence:** Suppose $|\lambda_1| > |\lambda_2| \geq |\lambda_i| > 0$ $\forall$ $i > 1$ and $y_1 = [V^{-1}v^{(0)}]_1 \neq 0$ ← first row of V^{-1} is not $\perp$ to $v^{(0)}$

then:

$$\|v^{(k)} - (\pm v_1)\| = \mathcal{O}\left(\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k\right) \quad \text{and} \quad |\lambda^{(k)} - \lambda_1| = \mathcal{O}\left(\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^{2k}\right)$$

($v^{(k)} \to v_1$, $\lambda^{(k)} \to \lambda_1$ geometrically at rates $|\lambda_2|/|\lambda_1|$ and $(|\lambda_2|/|\lambda_1|)^2$

errors converge linearly, $\|v^{(k)} - (\pm v_1)\| \leq C_k \|v^{(k-1)} - (\pm v_1)\|$ for $C_k \to |\lambda_2|/|\lambda_1|$)

• so, $|\lambda_2|/|\lambda_1|$ = ratio of largest eigenvalues determines the convergence rate...

• Why?

recall: $A^k x = V\Lambda^k V^{-1} x = \sum_{j=1}^{n} \lambda_j^k \overbrace{(V^{-1}x)_j}^{\text{"}y_j\text{"}} v_j$

$v^{(k)} \propto A^k v^{(0)}$

so $v^{(k)} \propto \sum_{j=1}^{n} \lambda_j^k (V^{-1}v^{(0)})_j v_j = \lambda_1^k \Big(\underset{\uparrow\,[V^{-1}v^{(0)}]_1}{y_1} v_1 + \underbrace{\sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_1}\right)^k y_j v_j}_{\text{component on span}(\{v_j\}_{j\neq 1})\text{ converges to 0 at rates }(\lambda_j/\lambda_1)} \Big)$

• **Problems:** 1. converges slowly if $|\lambda_2| \approx |\lambda_1|$
2. only finds (v_1, λ_1), the largest eigenpair...

• **Q:** how can we find a specific eigenpair?

- <u>idea</u>: matrix functions... if $f(x)$ is an analytic function $\mathbb{C} \to \mathbb{C}$

s.t.

$$f(x) = \sum_{j=0}^{\infty} \alpha_j x^j \quad \text{(power series)}$$

then

$$f(A) \equiv \sum_{j=0}^{\infty} \alpha_j A^j$$

if A is diagonalizable then $A = V \Lambda V^{-1}$, $A^j = V \Lambda^j V^{-1}$

so:

$$f(A) = \sum_{j=0}^{\infty} \alpha_j V \Lambda^j V^{-1} = V \left[\sum_{j=0}^{\infty} \alpha_j \Lambda^j \right] V^{-1} = V f(\Lambda) V^{-1}$$

$$= V \operatorname{diag}(f(\lambda_1), f(\lambda_2), \dots f(\lambda_n)) V^{-1}$$

- pick f to highlight a particular eigenvalue
- <u>Ex</u>: $f(x) = (x - \mu)^{-1}$ for some $\mu \in \mathbb{C}$

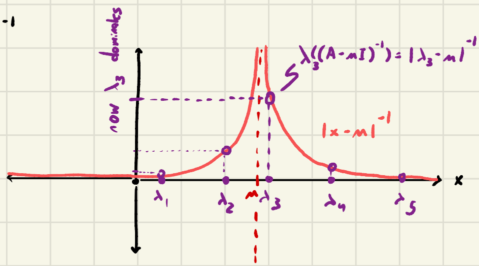
then: $f(A) = V \operatorname{diag}((\lambda_1 - \mu)^{-1}, (\lambda_2 - \mu)^{-1}, \dots (\lambda_n - \mu)^{-1}) V^{-1}$

$$= [V \operatorname{diag}(\lambda_1 - \mu, \lambda_2 - \mu, \dots \lambda_n - \mu) V^{-1}]^{-1}$$

$$= [V(\Lambda - \mu I) V^{-1}]^{-1}$$

$$= [V \Lambda V^{-1} - \mu I]^{-1}$$

$$= [A - \mu I]^{-1}$$



then:

$(A - \mu I)^{-1}$ has eigenvalues $\dfrac{1}{(\lambda_j - \mu)}$ provided $\mu \neq \lambda_j \ \forall j$

so, $A - \mu I$ is invertible if $\mu \neq \lambda_j \ \forall j$

and

$$\max_j \{ |\lambda_j((A - \mu I)^{-1})| \} = \max_j \left\{ \frac{1}{|(\lambda_j(A) - \mu)|} \right\} = \Big(\min_j \{ \underbrace{|\lambda_j(A) - \mu|}_{\text{eigenvalue closest to } \mu} \} \Big)^{-1}$$

- notice, we can apply power iteration to $(A - \mu I)^{-1}$ by implicitly multiplying by $(A - \mu I)^{-1}$ ... i.e. solving

$$(A - \mu I) v^{(k)} \propto v^{(k-1)}$$

$$\Downarrow \text{ implies}$$

$$v^{(k)} \propto (A - \mu I)^{-1} v^{(k-1)}$$

... leads to <u>inverse iteration</u>

2. Inverse Iteration:

1. input A, $v^{(0)}$ w/ $\|v^{(0)}\|_2 = 1$, μ ← guess at desired eigenvalue
2. iterate until stopping

   *(i) solve $(A - \mu I)w = v^{(k-1)}$ • apply $(A - \mu I)^{-1}$... raise the power

   (ii) $v^{(k)} = w/\|w\|$ • normalize → estimate eigenvector

   (iii) $\lambda^{(k)} = r(A, v^{(k)}) = v^{(k)*}(A v^{(k)})$ • estimate eigenvalue

* (i) is expensive, cost of solving linear system against $(A - \mu I)$

... but, if reduce $A - \mu I \to LU$ or $R^* R$ (cost: $O(n^3)$)

on 1st iteration, then on later iterations just use substitution

(cost $O(n^2)$, same as applying A...)

so, really only a 1-time cost.

• Convergence: power iteration w/ eigenvalues $(\lambda_j(A) - \mu)^{-1}$...

• Suppose $|\underset{\substack{\uparrow \\ \text{closest to } \mu}}{\lambda_J} - \mu| < |\underset{\substack{\uparrow \\ \text{next closest to } \mu}}{\lambda_K} - \mu| \le |\lambda_i - \mu| \;\; \forall\, i \ne J$ or K and $Y_J = [V^{-1} v^{(0)}]_J \ne 0$

then:

$$\|v^{(k)} - (\pm v_J)\| = O\left(\left(\frac{|\mu - \lambda_J|}{|\mu - \lambda_K|}\right)^k\right) \quad \text{and} \quad |\lambda^{(k)} - \lambda_J| = O\left(\left(\frac{|\mu - \lambda_J|}{|\mu - \lambda_K|}\right)^{2k}\right)$$

• Converges quickly if $\mu \approx \lambda_J$ and is much closer to λ_J than any other eigenvalue

• Can find (v_J, λ_J) s.t. $|\lambda_J - \mu|$ is minimized (λ_J closest to μ)

• Q: how to pick μ? Can we update w/ a better guess to an eigenvalue as we go?

• yes! use $\mu = \lambda^{(k)}$...

3. Rayleigh Quotient Iteration:

1. input A, $v^{(0)}$ w/ $\|v^{(0)}\|_2 = 1$,

2. iterate until stopping

(iii) $\lambda^{(k)} = r(A, v^{(k)}) = v^{(k)^*}(A v^{(k)})$ · estimate eigenvalue

*(i) solve $(A - \lambda^{(k)} I) w = v^{(k)}$ · apply $(A - \mu I)^{-1}$... raise the power

(ii) $v^{(k+1)} = w / \|w\|$ · normalize → estimate eigenvector

* now "(i)" is expensive... $\lambda^{(k)}$ changes w/ every step so need a new reduction everytime... cost $\mathcal{O}(n^3)$ if use direct methods

· Convergence: Rayleigh quotient iteration converges to some eigenvalue/vector pair for almost all $v^{(0)}$. Denote the pair (v_J, λ_J). Then, in the limit of large k, convergence is cubic ← !!! crazy fast: error $1/10 \to 1/10^3 \to 1/10^9 \to 1/10^{27}$ in the sense:

$$\|v^{(k+1)} - (\pm v_J)\| = \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^3)$$

$$|\lambda^{(k+1)} - \lambda_J| = \mathcal{O}(|\lambda^{(k+1)} - \lambda_J|^3)$$

· Why? b/c when $v^{(k)} \approx v_J$ then $\lambda^{(k)} = r(A, v^{(k)}) \approx \lambda_J$ w/ quadratic errors

$$(\lambda^{(k)} - \lambda_J = \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^2)$$

then the inverse iteration update to go to $v^{(k+1)}$ is linear:

$$\|v^{(k+1)} - (\pm v_J)\| = \mathcal{O}\left(\left(\frac{|\lambda_J - \lambda^{(k)}|}{|\underbrace{\lambda_L}_{\text{next closest}} - \lambda^{(k)}|}\right) \|v^{(k)} - (\pm v_J)\|\right)$$

where $|\lambda_J - \lambda^{(k)}| \to \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^2)$

$|\lambda_L - \lambda^{(k)}| \to \min_{\ell \neq J} \{|\lambda_\ell - \lambda_J|\} = \mathcal{O}(1)$

$$= \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^3)\,!$$

Thursday - May 11th - Eigenvalue Methods

- **Logistics:**
  - HW 5 due tonight
  - Project 2 posted, due Friday of finals week

- **Goals:**
  - Rayleigh quotient iteration
  - Simultaneous iteration
  - The QR algorithm
    - Shifts
    - Preprocessing (reduction to upper Hessenberg)

- **recall from Tuesday:**

1. Rayleigh quotient $r(A, v) = \dfrac{v^*Av}{v^*v}$

   is quadratically accurate $\approx$ to λ_j if $v \approx v_j$

2. Power iteration: $A^k v^{(0)} \propto v^{(k)} \longrightarrow v_1$ if $|\lambda_1| > |\lambda_j|$

   $\forall\ j > 1$ and converges linearly,

   $$\|v^{(k+1)} - (\pm v_1)\| = O\left(\frac{|\lambda_2|}{|\lambda_1|} \|v^{(k)} - (\pm v_1)\|\right)$$

   so

   second largest eigenvalue

   $$\|v^{(k)} - (\pm v_1)\| = O\left(\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k\right)$$

3. Inverse iteration:

   closest to μ

   $(A - \mu I)^{-k} v^{(0)} \propto v^{(k)} \longrightarrow v_J$ where $|\lambda_J - \mu| < |\lambda_L - \mu| \leq |\lambda_i - \mu|$

   $\forall\ i, L \neq J$

   next closest

   converges linearly,

   $$\|v^{(k+1)} - (\pm v_J)\| = O\left(\frac{|\lambda_J - \mu|}{|\lambda_L - \mu|} \|v^{(k)} - (\pm v_J)\|\right)$$

   so

   $$\|v^{(k)} - (\pm v_J)\| = O\left(\left(\frac{|\lambda_J - \mu|}{|\lambda_L - \mu|}\right)^k\right)$$

   fast if μ much closer to one eigenvalue than any of the others...

- **idea:** can we use $r(A, v^{(k)})$ to iteratively improve μ?

3. Rayleigh Quotient Iteration:

1. input A, $v^{(0)}$ w/ $\|v^{(0)}\|_2 = 1$,

2. iterate until stopping

(iii) $\lambda^{(k)} = r(A, v^{(k)}) = v^{(k)*}(Av^{(k)})$ • estimate eigenvalue

*(i) solve $(A - \lambda^{(k)} I) w = v^{(k)}$ • apply $(A - \mu I)^{-1}$... raise the power

(ii) $v^{(k+1)} = w/\|w\|$ • normalize → estimate eigenvector

* now "(i)" is expensive... $\lambda^{(k)}$ changes w/ every step so need a new reduction everytime... cost $\mathcal{O}(n^3)$ if use direct methods

• Convergence: Rayleigh quotient iteration converges to some eigenvalue/vector pair for almost all $v^{(0)}$.
Denote the pair (v_J, λ_J). Then, in the limit of large K, convergence is cubic ← !!! crazy fast: error $1/10 \to 1/10^3 \to 1/10^9 \to 1/10^{27}$
in the sense:

$$\|v^{(k+1)} - (\pm v_J)\| = \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^3)$$
$$|\lambda^{(k+1)} - \lambda_J| = \mathcal{O}(|\lambda^{(k)} - \lambda_J|^3)$$

• Why? b/c when $v^{(k)} \approx v_J$ then $\lambda^{(k)} = r(A, v^{(k)}) \approx \lambda_J$ w/ quadratic errors

$$(\lambda^{(k)} - \lambda_J = \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^2)$$

then the inverse iteration update to go to $v^{(k+1)}$ is linear:

$$\|v^{(k+1)} - (\pm v_J)\| = \mathcal{O}\left(\left(\frac{|\lambda_J - \lambda^{(k)}|}{|\lambda_L - \lambda^{(k)}|}\right) \|v^{(k)} - (\pm v_J)\|\right)$$

↑ next closest

where $|\lambda_J - \lambda^{(k)}| \to \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^2)$

$|\lambda_L - \lambda^{(k)}| \to \min_{\ell \neq J} \{|\lambda_\ell - \lambda_J|\} = \mathcal{O}(1)$

$$= \mathcal{O}(\|v^{(k)} - (\pm v_J)\|^3)\,!$$

• Rayleigh quotient iteration finds one eigenpair... how do we find all of the eigenpairs at once?

• idea: run iteration on multiple input directions at once...

• from now on, assume A is diagonalizable

- **Simultaneous Iteration:** let's run power iteration on many inputs simultaneously...

  - **idea:** input $V^{(0)} = \begin{bmatrix} | & | & & | \\ v_1^{(0)} & v_2^{(0)} & \cdots & v_\ell^{(0)} \\ | & | & & | \end{bmatrix}$ ℓ initial guesses to eigenvectors

    iterate: $W = AV^{(k)}$

    $$V^{(k+1)} = \begin{bmatrix} | & | & & | \\ w_1/\|w_1\| & w_2/\|w_2\| & \cdots & w_\ell/\|w_\ell\| \\ | & | & & | \end{bmatrix}$$

    then $\|v_j^{(k)}\| = 1$ and $v_j^{(k)} \propto A^k v_j^{(0)}$

  - **problem:** all $v_j^{(k)} \longrightarrow v_1$ (dominant eigenvector)

    since

    $$\frac{1}{|\lambda_1|^k} A^k = V \begin{bmatrix} 1 & & & \\ & (\lambda_2/\lambda_1)^k & & \\ & & \ddots & \\ & & & (\lambda_n/\lambda_1)^k \end{bmatrix} V^{-1} \xrightarrow[k\to\infty]{} V \begin{bmatrix} 1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix} V^{-1}$$

    <1 <1

  - **soln:** if $v_1^{(k)} \longrightarrow v_1$ then, by keeping $v_{j\neq 1}^{(k)}$ "away" from $v_1^{(k)}$, we hope $v_{j\neq 1}^{(k)}$ will converge to a different eigenvector.

- Suppose A is Hermitian ($A = A^*$). Then A is unitarily diagonalizable...

  $$A = V\Lambda V^* \text{ where } V \text{ is unitary}$$

  ($v_i \perp v_j \ \forall\ i\neq j$ and $\|v_i\| = 1 \ \forall\ i$)

- then, since all the eigenvectors are $\perp$, let's keep all of the iterates $v_j^{(k)}$ $\perp$ to one another...

  if $v_i^{(k)} \perp v_j^{(k)}$ $\forall$ $i\neq j$ and k then

  $v_i^{(k)} \perp v_1^{(k)}$, $v_1^{(k)} \longrightarrow v_1$ so $v_i^{(k)} \longrightarrow \in \mathbb{R}^n / v_1 = \operatorname{span}(v_1)^\perp$

  that is $\lim_{k\to\infty} v_{i\neq 1}^{(k)} \in \{ w \in \mathbb{R}^n \mid w \perp v_1 \}$

- in particular, if we maintain $\perp$ in a triangular fashion:

  $$v_1^{(k)} \propto A^k v_1^{(0)}, \quad v_2^{(k)} \propto \left(A^k v_2^{(0)}\right)_{\perp v_1^{(k)}}, \ \ldots \quad v_j^{(k)} \propto \left(A^k v_j^{(0)}\right)_{\perp \operatorname{span}(v_1^{(k)}, \ldots, v_{j-1}^{(k)})}$$

then, if $|\lambda_1| > |\lambda_2| > |\lambda_3| > \dots$

$$v_1^{(k)} \longrightarrow v_1$$

so $\quad v_2^{(k)} \longrightarrow \left(A^k v_2^{(0)}\right)_{\perp v_1} \longrightarrow v_2$ (eigenvector for next largest eigenvalue)

$\vdots$

so $\quad v_j^{(k)} \longrightarrow \left(A^k v_j^{(0)}\right)_{\perp \operatorname{span}(v_1, \dots v_{j-1})} \longrightarrow v_j$

- can prove inductively... if $v_i^{(k)} \longrightarrow v_i \;\; \forall \; i \le j$

then

$$v_{j+1}^{(k)} \propto \left[A^k v_{j+1}^{(0)}\right]_{\perp \operatorname{span}\{v_1^{(k)}, \dots v_j^{(k)}\}} = \left[\sum_{i=1}^{n} \lambda_i^k \left(v_i^* v_{j+1}^{(0)}\right) v_i\right]_{\perp \operatorname{span}\{v_1^{(k)}, \dots v_j^{(k)}\}}$$

$$= \sum_{i=1}^{n} \lambda_i^k \left(v_i^* v_{j+1}^{(0)}\right) \underbrace{v_{i\,\perp \operatorname{span}\{v_1^{(k)}, \dots v_j^{(k)}\}}}$$

$v_\ell^{(k)} \to v_\ell$ as $k \to \infty$ for $\ell \le j$

so $v_{i\,\perp \operatorname{span}\{\dots\}} \to 0$ if $i \le j$, $\to v_i$ if $i > j$ since v's are $\perp$

$$\longrightarrow \lambda_{j+1}^k \left(v_{j+1}^* v_{j+1}^{(0)}\right) v_{j+1} + \sum_{i > j+1} \lambda_i^k \left(v_i^* v_{j+1}^{(0)}\right) v_i + \text{vanishing projection onto first } j \text{ eigenvectors}$$

dominates $\qquad |\lambda_i| < |\lambda_{j+1}|$

so:

$$v_{j+1}^{(k)} \longrightarrow v_{j+1}.$$

- leads to an algorithm:

- <u>Simultaneous Iteration</u>:

# of desired eigenvalues

1. input $A \in \mathbb{C}^{n \times n}$ Hermitian, $V^{(0)} \in \mathbb{C}^{n \times \ell}$
2. orthogonalize $V^{(0)} \longrightarrow Q^{(0)}$
3. iterate for $k = 1, 2, \dots$ until stopping
   - (i) $W = A Q^{(k-1)}$ ← increase the power
   - (ii) $Q^{(k)} R^{(k)} = W$ ← triangular orthogonalization (use Householder)
   - (iii) $\lambda_j^{(k)} = r(A, q_j^{(k)}) = q_j^{(k)*}(A q_j^{(k)})$ ← optional

then $q_j^{(k)}$ are normalized and mutually $\perp$ and converge to the ℓ largest eigenvectors $v_j \longrightarrow \underbrace{\lambda_j^{(k)} = r(A, q_j^{(k)})} \xrightarrow{k \to \infty} \lambda_j$

use Rayleigh to estimate eigenvalues...

- simultaneous iteration is usually expressed in a different order...

orthogonalize, then raise the power (multiply by A)

- **QR Iteration:**
  1. input $A \in \mathbb{C}^{n \times n}$, Hermitian, $A^{(0)} = A$
  2. iterate for $k = 1, 2, \dots$ until stopping

all backward stable

(i) $Q^{(k)} R^{(k)} = A^{(k)}$ ⟵ orthogonalize (triangular), use Householder — cost $\mathcal{O}(4/3 n^3)$

(ii) $A^{(k+1)} = R^{(k)} Q^{(k)}$ ⟵ implicitly multiply by A — cost $\mathcal{O}(n^3)$

$\Big\}\ \mathcal{O}(\frac{10}{3} n^3)$

(iii) $\lambda_j^{(k)} = r(A, q_j^{(k)}) = q_j^{(k)*} A\, q_j^{(k)} = A_{jj}^{(k+1)}$ ⟵ optional

cost: $\mathcal{O}(Kn^3)$!!!

- are these the same?
  - let $Q_s^{(k)}, R_s^{(k)}$ be the Q's and R's produced by simultaneous iteration initialized w/ $V^{(0)} = I$.
  - let $Q^{(k)}, R^{(k)}$ be the Q's and R's produced by QR iteration initialized w/ $V^{(0)} = I$.
  - let A^k be the k^{th} power of A
  - then:

$A^{(k)}$ and A^k

(i) $R_s^{(k)} = R^{(k)}$

(ii) $\underbrace{Q_s^{(k)} = \prod_{j=1}^{k} Q^{(k)} = Q^{(1)} Q^{(2)} \dots Q^{(k)}}_{R\text{'s and } Q\text{'s}}$

(iii) $A^{(k)} = Q_s^{(k)*} A\, Q_s^{(k)}$

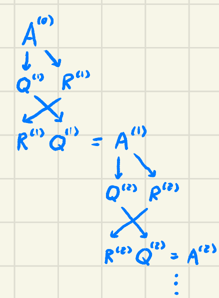
(iv) $A^k = Q_s^{(k)} \underbrace{\left[R_s^{(k)} R_s^{(k-1)} \dots R_1^{(1)} \right]}_{= R^{(k)} R^{(k-1)} \dots R^{(1)}}$

implicitly, QR of A^k

- let's check (iii)
  - $A^{(0)} = A$
  - $Q^{(1)} R^{(1)} = A^{(0)} \longmapsto R^{(1)} = Q^{(1)*} A^{(0)}$
  - $A^{(1)} = R^{(1)} Q^{(1)} = Q^{(1)*} A^{(0)} Q^{(1)} = Q^{(1)*} A\, Q^{(1)}$
  - $Q^{(2)} R^{(2)} = A^{(1)} \longmapsto R^{(2)} = Q^{(2)*} A^{(1)}$
  - $A^{(2)} = R^{(2)} Q^{(2)} = Q^{(2)*} A^{(1)} Q^{(2)} = (Q^{(2)*} Q^{(1)*}) A^{(0)} (Q^{(1)} Q^{(2)})$

$\vdots \qquad = (Q^{(1)} Q^{(2)})^* A\, (Q^{(1)} Q^{(2)})$

  - $A^{(k)} = Q^{(k)*} A^{(k-1)} Q^{(k)}$



so:

$$A^{(k)} = (Q^{(1)} Q^{(2)} \dots Q^{(k)})^* A\, (Q^{(1)} Q^{(2)} \dots Q^{(k)}) = Q_s^{(k)*} A\, Q_s^{(k)} \ ✓$$

• Equation (iii): $A^{(k)} = Q_s^{(k)*} A\, Q_s^{(k)}$

where $Q_s^{(k)} = Q^{(1)} Q^{(2)} \dots Q^{(k)}$

provides an alternative perspective on QR iteration...

• <u>observations</u>: 1. $Q_s^{(k)}$ is unitary so

$$Q_s^{(k)*} = Q_s^{(k)^{-1}}$$

$$\therefore \quad A^{(k)} = Q_s^{(k)^{-1}} A\, Q_s^{(k)} \iff Q_s^{(k)} A^{(k)} Q_s^{(k)^{-1}} = A$$

are similarity transforms.

$\therefore$ $A^{(k)}$ and A have the same eigenvalues

2. $$\lambda_j^{(k)} = r(A, q_{s_j}^{(k)}) = q_{s_j}^{(k)*} A\, q_{s_j}^{(k)} = e_j^T \left[Q_s^{(k)*} A\, Q_s^{(k)} \right] e_j = A_{jj}^{(k)}$$

and $\lambda_j^{(k)} \longrightarrow \lambda_j$ so $A_{jj}^{(k)} \to \lambda_j$

... the diagonal entries of $A_{jj}^{(k)}$ converge to the eigenvalues of A.

3. $q_{s_j}^{(k)} \longrightarrow v_j$ so $Q_s^{(k)} \longrightarrow V$ so, if $A = V \Lambda V^{-1}$

then $A^{(k)} = Q_s^{(k)*} A\, Q_s^{(k)} \longrightarrow V^{-1} V \Lambda V^{-1} V = \Lambda$

from convergence of simultaneous iteration

so:

$$Q_s^{(k)} = Q^{(1)} Q^{(2)} \dots Q^{(k)} \longrightarrow V$$

$$A^{(k)} = Q_s^{(k)*} A\, Q_s^{(k)} \longrightarrow \Lambda.$$

$(A_{jj}^{(k)} \to \lambda_j,\ A_{ij}^{(k)} \to 0 \text{ if } i \neq j)$

QR iteration is an <u>eigenvalue revealing</u> <u>iteration.</u> At each stage we implicitly perform a unitary similarity transform of A that takes $A^{(k)} \to A^{(k+1)} = Q^{(k+1)*} A^{(k)} Q^{(k+1)}$ s.t. as $k \to \infty$, $A^{(k)} \to \Lambda$

and, the product of the transforms: $Q_s^{(k)} = Q^{(1)} Q^{(2)} \dots Q^{(k)}$ converges to V, $Q_s^{(k)} \longrightarrow V$

• Convergence follows from convergence of power iteration,

if A Hermitian, $|\lambda_1| > |\lambda_2| > \dots |\lambda_n|$ then $A^{(k)} \to \Lambda$,

and $Q_s^{(k)} \to V$ geometrically (linear convergence) w/ constant:

$$C = \max_{j \in [1, n-1]} \left\{ |\lambda_{j+1}| / |\lambda_j| \right\}$$

(if use $V^{(0)} \in \mathbb{C}^{n \times \ell}$ then $C = \max_{j \in [1, \ell-1]} \{ \dots \}$)

slow for similar eigenvalues...

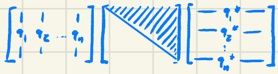
- Combining ideas from:
  - (i) eigenvalue revealing iteration → use to preprocess, reduce cost of each QR calculation
  - (ii) Rayleigh quotient iteration (shifting) → use to accelerate convergence, isolate particular eigenvalues, then split into smaller subproblems (see lectures 28 and 29)

  gives a practical (widely used) algorithm...

- <u>Eigenvalue Revealing Factorizations (and Iteration)</u>:

  - <u>Def</u>: given any $A \in \mathbb{C}^{n\times n}$ then $\exists$ a <u>Schur</u> <u>decomposition</u> of A:

    $$A = Q\,T\,Q^*$$ =

    where:

    (i) $Q \in \mathbb{C}^{n\times n}$ is unitary

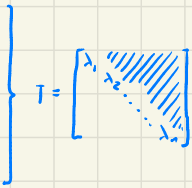
    (ii) T is upper triangular

  - <u>Facts</u>: 1. $Q^* = Q^{-1}$ so $Q\,T\,Q^* = A$, $T = Q^* A Q$ are similarity transforms so $\Lambda(T) = \Lambda(A)$ (A and T have the same eigenvalues)

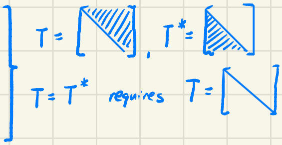
    2. T is triangular so $\lambda_j(A) = \lambda_j(T) = T_{jj}$

       ... the eigenvalues of A are the diagonal entries of T.

       hence T is "eigenvalue revealing"

    3. if A is Hermitian, $T = Q A Q^*$ is Hermitian, all Hermitian triangular matrices are diagonal so T is diagonal, $T = \Lambda$ so $Q = V$.

- <u>Eigenvalue revealing iteration</u>: find Q s.t. $A = Q\,T\,Q^* \Rightarrow T = Q^* A Q$

  impossible w/ a direct method (otherwise, direct methods for eigenvalues would exist)

  - instead, attempt iteratively: $A^{(k)} = Q^{(k)*} A^{(k-1)} Q^{(k)}$

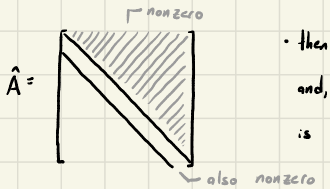
    s.t. $A^{(k)} \to T$

    and $Q_s^{(k)} = Q^{(1)} Q^{(2)} \cdots Q^{(k)} \longrightarrow Q$

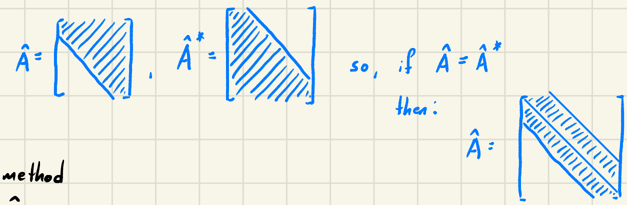
  - <u>exactly</u> what QR iteration does. Stable since all operations are unitary.

- **Preprocessing**: before starting QR iteration find a unitary similarity transform of A, $\hat{A} = \hat{Q}^* A \hat{Q}$ s.t. $\hat{A}$ is cheaper to work with (want $\hat{A}$ sparse, $\hat{A} \approx T$)
  - usually, aim for $\hat{A}$ upper Hessenberg...

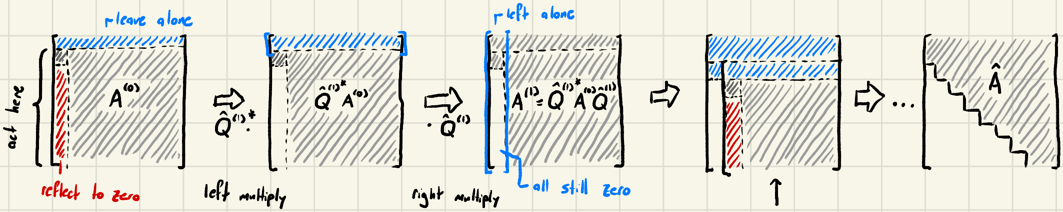
- **Def**: $\hat{A} \in \mathbb{C}^{n \times n}$ is upper Hessenberg if $\hat{A}_{ij} = 0 \ \forall \ i > j+1$.
  (upper triangular + 1 nonzero subdiagonal)



- then $\hat{A} \approx T$ (where nonzero, just 1 extra off-diagonal) and, if A is Hermitian, $\hat{A}$ is Hermitian so $\hat{A}$ is tridiagonal thus sparse



- Can reduce from $A \to \hat{A}$ w/ a direct method
(see lecture 26, use Householder to form $\hat{Q}$ recursively...



- cost: $\mathcal{O}(\frac{10}{3} n^3)$ generically, $\mathcal{O}(\frac{4}{3} n^3)$ if Hermitian
- backward stable (unitary operations / Householder)

- using $\hat{A}$ cuts cost of each step of QR iteration
- if A is Hermitian $\Rightarrow$ $\hat{A}$ is tridiagonal
reduces cost of each QR step to $\mathcal{O}(n^2)$
so cost of QR iteration is $\mathcal{O}(\#\text{ iterations} \cdot n^2)$
(often # iterations $\ll n$ ... reduction to upper Hessenberg more expensive than subsequent iteration!)

- **Shifting**: update QR step w/ shift $\mu^{(k)}$...

$$\text{(i)}\quad Q^{(k)} R^{(k)} = A^{(k-1)} - \mu^{(k)} I$$
$$\text{(ii)}\quad A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$$

where $\mu^{(k)}$ is chosen to speed convergence

($\underline{\text{Ex}}$: $\mu^{(k)} = A_{jj}^{(k-1)} = r(A, q_{s_j}^{(k-1)}) = \lambda_j^{(k)} \simeq \lambda_j$)

- **Method**: 1. use Householder to reduce to upper Hessenberg
  2. run QR iteration w/ shifts

  - achieves: cubic convergence like Rayleigh quotient iteration

    produces backward stable estimates

$$\tilde{V} = Q_s^{(k)} \simeq V$$
$$\tilde{\Lambda} = \text{diag}(A^{(k)}) \simeq \Lambda$$

s.t.

$$\tilde{A} = \tilde{V} \tilde{\Lambda} \tilde{V}^{-1} \quad \text{satisfies} \quad \frac{\|\tilde{A} - A\|}{\|A\|} = \mathcal{O}(\varepsilon_m)$$

and

$$\frac{|\tilde{\lambda}_j - \lambda_j|}{\|A\|} = \mathcal{O}(\varepsilon_m).$$

- **Warning**: conditioning of the eigenvectors & values of A in perturbations to A can be very bad, especially if A is not unitarily diagonalizable!
  - stability of (λ_j, v_j) depends on gap between λ_j and next nearest eigenvalue...

Week 9 - Applications of Spectral Linear Algebra: Embedding Data in Low-Dimensions

Tuesday - May 16^{th}

- **Logistics:**
  - HW 6 and Project 2 posted, due the 25^{th} and 26^{th}

- **Learning Goals:**
  1. State an embedding/low-dimensional data representation problem
  2. Relate spectral linear algebra to geometry of data
  3. Perform PCA using the SVD and explain why it works

An Embedding Problem:

- Suppose we are given a collection of data vectors

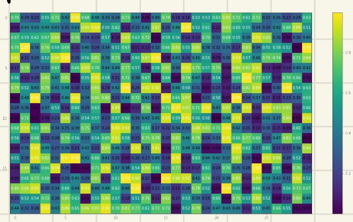
$$\{x_1, x_2, \dots x_m\}, \quad x_j = \begin{bmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{bmatrix} \in \mathbb{R}^n$$

where x_{ij} is the i^{th} quantity measured in the j^{th} trial/subject/example.

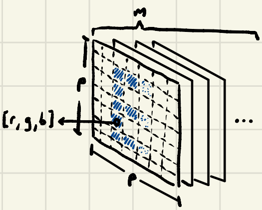
- collect the data into a matrix X:

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x_{:1} & x_{:2} & \cdots & x_{:m} \\ | & | & & | \end{bmatrix}}_{m} \Big\updownarrow n = \begin{array}{c} \\ i \end{array} \begin{array}{c} \text{trial/example} \\ \begin{bmatrix} & & \\ \text{———} & x_{ij} & \text{———} \\ & & \end{bmatrix} \\ j \end{array} \text{quantity measured}$$

"X"



- **Ex:** collection of m images, each containing p^2 pixels, color of each pixel is determined by 3 #'s (r, g, b)



then $x_{:j} \in \mathbb{R}^{3p^2}$, m vectors x

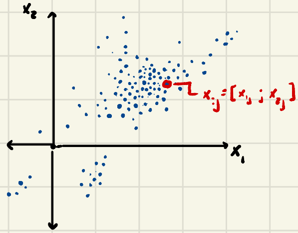
- Classic Ex: collection of hand drawn digits pictures of faces, pictures of cats and dogs

(MNIST: $m = 60 \times 10^4$ images of hand-drawn digits)

- **Ex:** genotype data: $m = 1,387$ individuals (from pool of 3×10^3 European study participants)

  $n = 197,146$ loci

- Each data vector $x \in \mathbb{R}^n$ can be treated as a point in an n-dimensional (n-D) space



- Collection of m points form a scatter cloud

- Problem: n is often large, or very large
  - Ex: a 480p image has $n = 1{,}353{,}600 = \mathcal{O}(10^6)$
    genetic data using $\mathcal{O}(10^5)$ loci

  million dimensional!

- high-D data can be difficult to work with:
  - memory intensive $\longrightarrow$ desire: compression
  - visualize $\longrightarrow$ desire: reduce to 2-4 dimensions

  reduce dimension...

  - interpret: individual entries are rarely meaningful alone
    meaning stored in the collection of values...
    - really an issue of basis:

$$x = x_1 \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \dots x_n \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

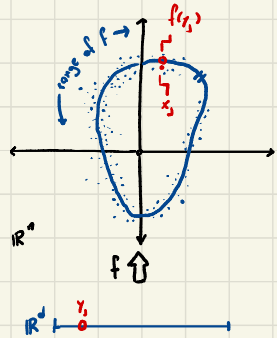
  rarely meaningful alone
  don't represent common features in the data

  $\longrightarrow$ desire: work in a meaningful coordinate system...
  use a basis that captures relevant features or patterns in data, separates distinct examples (for classification)

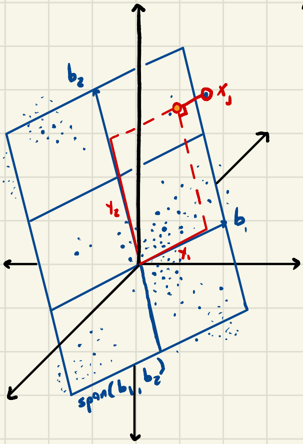
  ... meaningfully.

- Embedding Problem: given $X = \{x_{:1}, x_{:2}, \dots x_{:m}\}$, $x_j \in \mathbb{R}^n$ find a mapping that sends each data vector $x_j \longrightarrow y_j \in \mathbb{R}^d$ where $d \ll n$ that preserves relevant patterns/structure in X
  - aim to lose as little info about X as possible
  - often seek a mapping s.t. the entries of $y \in \mathbb{R}^d$ (directions in $\mathbb{R}^d$) are intrinsically meaningful...
    geometry in the embedding/latent space $\longrightarrow$ meaning in data

- Representation in a latent space: seek a mapping $f: \mathbb{R}^d \to \mathbb{R}^n$ s.t. $x_j \approx f(y_j)$ usually for $d \ll n$.
  - then, data X is concentrated near a manifold equal to the range of f



- **Linear Representation:** $f(y)$ is an affine function of y.
  that is: $f(y) = \bar{x} + By$, $\bar{x} \in \mathbb{R}^n$, $B \in \mathbb{R}^{n \times d}$



- represents data as a linear combination of d basis vectors $b_{:1}, b_{:2}, \dots b_{:d}$ where $b_{:j} \in \mathbb{R}^n$
  - basis vectors $\{b_{:j}\}_{j=1}^{d}$ are "feature vectors", represent x as a weighted combination of features, weights $\{y_j\}_{j=1}^{d}$
  - the range of f is the affine subspace $\bar{x} \oplus \text{range}(B)$ possible if X concentrated near a d dimensional subspace
  - for fixed $\bar{x}$, B find y_j from x_j by solving the LS problem:

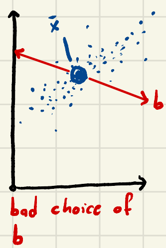
    minimize: $\| By - (x_j - \bar{x}) \|^2$ over all $y \in \mathbb{R}^d$.

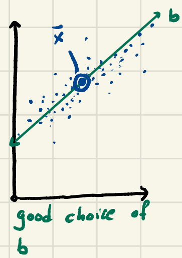
    so, solve by projection: $y_j = (x_j - \bar{x})_{\parallel \text{range}(B)}$

- Can find y from x, B, $\bar{x}$ via projection...

  - how to choose $(B, \bar{x})$? ← a new problem...

  - pose as an optimization problem:
    - find a d-dimensional subspace, range$(B) + \bar{x}$ s.t. projection onto the subspace <u>retains most of the "information" or "structure" of X.</u>





    - what we mean by "information/structure" determines the form of the problem...

      - Ex: aim to maintain as much variance/spread in the data as possible
        - relevant to linear classifiers & regression, normally distributed data.

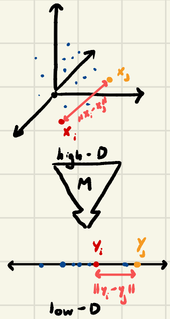
      - Ex: aim to maintain pairwise distances and angles between embedded data points...

- <u>Question</u>: Why can we hope to find a subspace of dimension $d \ll n$ s.t. projection onto it retains most of structure of X?

  - <u>Johnson-Lindenstrauss Lemma</u>:
    Given any X, $\{x_{:j}\}_{j=1}^{m}$, $x_{:j} \in \mathbb{R}^n$ let $0 < \varepsilon < 1$ and $d = \lceil 8 \ln(m)/\varepsilon^2 \rceil$.
    Then, there is a linear map $M: \mathbb{R}^n \rightarrow \mathbb{R}^d$ such that

$$(1-\varepsilon)\|x_i - x_j\|^2 \leq \|\underbrace{M(x_i - x_j)}_{y_i - y_j}\|^2 \leq (1+\varepsilon)\|x_i - x_j\|^2$$

    for all $1 \leq i, j \leq m$.



  - loosely: any set of points in a high-D space can be linearly mapped (say, projected) into a low-D space while nearly preserving the distance between points.
    <u>and</u> the low dimension d is logarithmic in the # of points m.

  - can extend to similar statements regarding the inner-products ... nearly preserve angles.

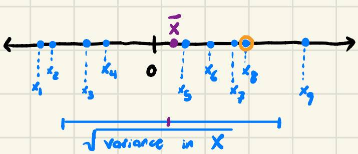
- So, can reasonably hope to find a linear representation s.t. projection retains most of the structure in X ...

- <u>Principal Component Analysis (PCA)</u>:
  - informally: given X, find a linear representation of X using shift $\bar{x}$ and features $Q = \{q_1, \dots q_d\}$ s.t

    1. features are independent ($q_i \perp q_j$ for $i \neq j$)

    2. projection onto $\bar{x} + \text{span}(Q)$ retains as much variance as possible.

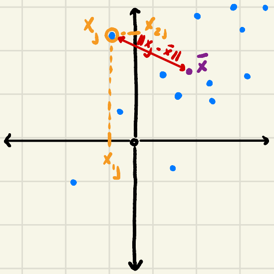
  - <u>Variance in data</u>: given samples $\{x_j\}_{j=1}^{m}$, $x_j \in \mathbb{R}$



    - <u>Centroid</u>: $\bar{x}$ = average of samples $= \frac{1}{m}\sum_{j=1}^{m} x_j$

    - <u>Sample Var in X</u> = average distance to $\bar{x}$ squared (when x_j drawn uniformly from X) $= \frac{1}{m-1}\sum_{j=1}^{m}(|x_j - \bar{x}|)^2$

- <u>n-dimensions</u>: given $\{x_j\}_{j=1}^m$, $x_j \in \mathbb{R}^n$



- <u>Centroid</u>: $\bar{x} = \frac{1}{m}\sum_{j=1}^m x_j$ so $\bar{x}_i$ = average value across i^{th} row of X

- <u>Component-Wise Variance</u>: Var in i^{th} comp $= \frac{1}{m-1}\sum_{j=1}^m (|x_{ij} - \bar{x}_i|)^2$

- <u>Variance</u>: sum of component-wise variances

$$Var[X] = \frac{1}{m-1}\sum_{i=1}^n \sum_{j=1}^m (|x_{ij} - \bar{x}_i|)^2 = \frac{1}{m-1}\sum_{j=1}^m \sum_{i=1}^n (|x_{ij} - \bar{x}_i|)^2$$
$$= \frac{1}{m-1}\sum_{j=1}^m \|x_j - \bar{x}\|^2$$
$$= \text{average (distance from randomly sampled } x_j \text{ to } \bar{x})^2$$

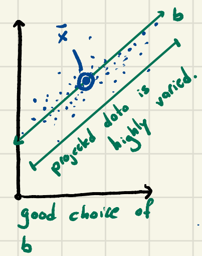
$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x_{:1} & x_{:2} & \cdots & x_{:m} \\ | & | & & | \end{bmatrix}}_{m} \Big\updownarrow n$$

compute average & var. across rows

sum down rows

- We've seen this before:
  - define centered data matrix x_j^c = original $x_j - \bar{x}$
  - <u>recall</u>: given $A \in \mathbb{R}^{n\times m}$, the <u>Frobenius norm</u> of A is $\|A\|_{Fro}^2 = \sum_{i,j} a_{ij}^2$
  - then:

$$Var[X^c] = \frac{1}{m-1}\sum_{i,j=1}^{n,m} (x_{ij}^c)^2 = \frac{1}{m-1}\|X^c\|_{Fro}^2$$



- So, goal is to embed X by projecting onto a d-dimensional subspace while maximizing the Frobenius norm of the projection...

- Assume X is centered (if not, replace X with X^c)
- Then, given d basis vectors $\{q_1, q_2, \ldots q_d\}$, all $\perp$ and normalized:
  - approximate:

$$x_j = \begin{bmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{nj} \end{bmatrix} \approx \sum_{k=1}^d y_{kj} \begin{bmatrix} | \\ q_k \\ | \end{bmatrix} = n\Big\updownarrow \overbrace{\begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_d \\ | & | & & | \end{bmatrix}}^{d} \begin{bmatrix} y_{1j} \\ y_{2j} \\ \vdots \\ y_{dj} \end{bmatrix} = Q^{(d)} y_j \quad \text{where} \quad Q^{(d)} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_d \\ | & | & & | \end{bmatrix}$$

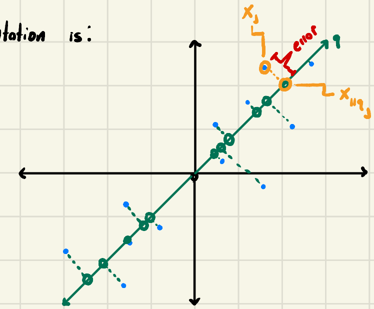
  - then:

$$X = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_m \\ | & | & & | \end{bmatrix} \approx \begin{bmatrix} | & | & & | \\ Q^{(d)}y_1 & Q^{(d)}y_2 & \cdots & Q^{(d)}y_m \\ | & | & & | \end{bmatrix} = Q^{(d)} \overbrace{\begin{bmatrix} | & | & & | \\ y_1 & y_2 & \cdots & y_m \\ | & | & & | \end{bmatrix}}^{m} \Big\updownarrow d = Q^{(d)} Y^{(d)}$$

  - so: $X \approx Q^{(d)} Y^{(d)}$ solve for $Y^{(d)}$ by projection: $Y^{(d)} = Q^{(d)^T} X$

$$X \approx \underbrace{Q^{(d)} Q^{(d)^T}}_{\perp \text{ projector onto range}(Q^{(d)})} X = P_{\parallel Q} X = X_{\parallel Q}$$

- If $Y^{(d)} = Q^{(d)^T} X$, $X \approx Q^{(d)} Y^{(d)}$ then the error in the representation is:



$$E^{(d)} = X - Q^{(d)} Y^{(d)} = X - X_{\parallel Q} = \underbrace{X_{\perp Q}}_{\text{component of } X \perp \text{ to range}(Q)}$$

so $$\|E^{(d)}\|_{Fro}^2 = \sum_{i,j} (x_{ij} - x_{\parallel Q_{ij}})^2 = m\underbrace{\left(\frac{1}{m}\sum_{j=1}^{m} \|x_j - x_{\parallel Q_j}\|^2\right)}_{\text{average distance from } x_j \text{ to its approximation}}$$

- PCA (formally): given $X \in \mathbb{R}^{n\times m}$ (centred) find $Q \in \mathbb{R}^{n\times d}$ s.t.
  1. independence: $q_i \perp q_j$ for any $i \neq j$, ($\|q_i\| = 1$ for any i)
  2. accuracy: set:
     $$Y = Q^T X, \quad X \approx Q Y, \quad E = X - QY$$
     maximize:
     $$\text{Var}[Y] \text{ given } Q \Rightarrow \text{maximize: } \|Q^T X\|_{Fro}^2$$

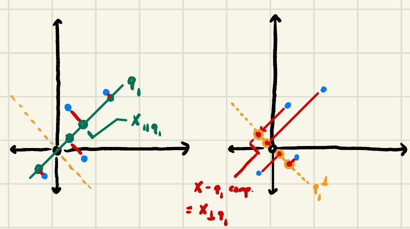
- Soln:
  1. center the data
  2. SVD: $X = U \Sigma V^T$
  3. set $q_j = u_j$.

Why?

- Claim: $q_j = u_j$ maximizes the variance after projection.
- Prove:
  - take a greedy approach (solve for q_1, then q_2, ...)
  - apply geometric interpretation of the SVD

- Original Problem: find $Q \in \mathbb{R}^{n\times d}$, orthonormal col's, maximizing $\|Q^T X\|_{Fro}^2$.

- "Greedy" Approach: solve for q_1 first, maximize $\|q^T X\|^2$
  subtract off projection of X onto q_1
  then solve for q_2 maximizing the var. remaining
  - iterate.



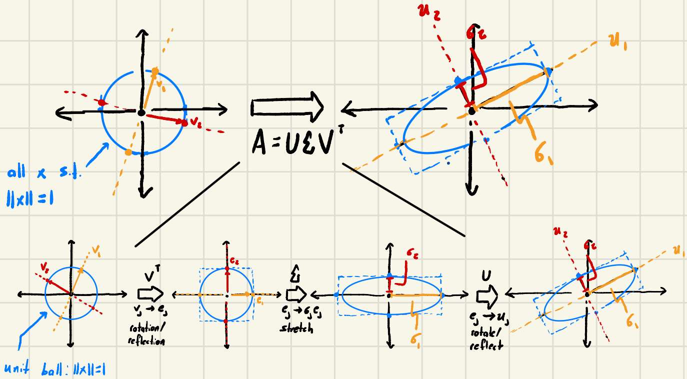
- Solving for q_1: given some q, $y = q^T X$
  - X is centered so $\text{mean}_j\{y_j\} = q^T \text{mean}\{X\} = 0$.

  thus,
  $$\text{var}(y) = \frac{1}{m-1}\sum_j^m (y_j)^2 = \frac{1}{m-1}\|y\|^2 = \frac{1}{m-1}\|q^T X\|^2 = \frac{1}{m-1}\|X^T q\|^2$$

  - so, to maximize $\text{var}(y)$, maximize $\|X^T q\|$ over all $\|q\| = 1$

to maximize $var(y)$ maximize $\|X^T q\|$ over all $\|q\|=1$...

- **recall, geometry of SVD:** given $A \in \mathbb{R}^{m \times n}$

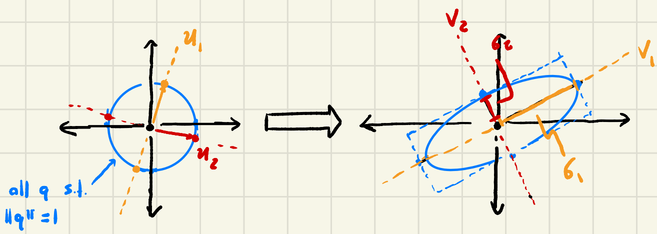


- so, A sends $\perp$-normal basis $\{v_1, v_2 \dots v_n\}$ to orthogonal basis $\{\sigma_1 u_1, \sigma_2 u_2, \dots \sigma_m u_m\}$
    - sends $v_j \longrightarrow \sigma_j u_j$ for $j \leq \min(m,n)$

    ($A v_j = \sigma_j u_j$ so, the input stretched the most by A is v_1 since $\sigma_1 \geq \sigma_j$ for $j > 1$)

- thus, given A: maximize $\|Aq\|$ over all q s.t. $\|q\|=1$ is solved by $q = v_1$, $\|A v_1\| = \sigma_1$.

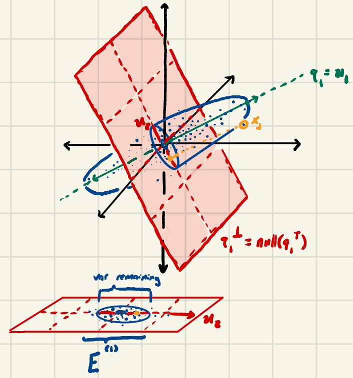
so, if $X = U\Sigma V^T$ then $X^T = V\Sigma^T U^T$ (swap roles of V and U)



the input $u_1 \to \sigma_1 v_1$ maximizes $\|X^T q\|$ over all $\|q\|=1$

thus, $q_1 = u_1$.

- What about the remaining q's?



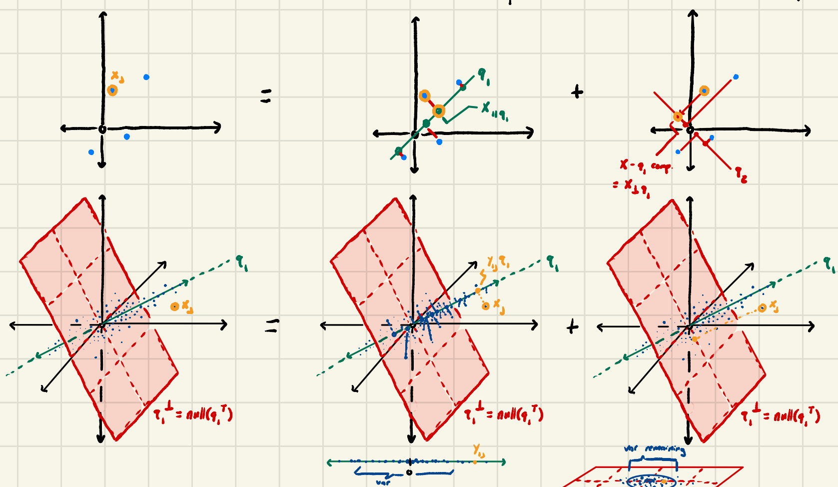
  - Induction:
    - show that, if $q_K = u_K$ for $K \leq j$ then $q_{j+1} = u_{j+1}$.
    - argue using projection. (see below)

$\downarrow$

Data: X = Representation: $X^{(1)} = X_{\parallel q_1}$ + Remainder: $E^{(1)} = X_{\perp q_1}$

2D:

3D:



Project X onto $Q^{(j)}$
maximize variance

Project X $\perp$ to $Q^{(j)}$
minimize variance

after projection: projected off

$$X_{\perp Q^{(j)}} = X - X_{\parallel Q^{(j)}} = U \begin{bmatrix} 0 & & & & \\ & \ddots & & & \\ & & \sigma_{j+1} & & \\ & & & \ddots & \\ & & & & \sigma_n \end{bmatrix} V^T = \begin{bmatrix} | & & | \\ u_{j+1} & \cdots & u_m \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_{j+1} & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix} \begin{bmatrix} - & v_{j+1}^T & - \\ & \vdots & \\ - & v_n^T & - \end{bmatrix}$$

... next largest singular value is σ_{j+1}
corresponds to vectors u_{j+1}, v_{j+1}.

- So, by induction: PCA is solved by the SVD! *
  1. $q_j = u_j$ for $j \in [1, d]$ $(d \leq rank(X))$ $\longrightarrow$ u's are $\perp$ so $q_i \perp q_j$ for $i \neq j$
  2. $$X^{(d)} = \underbrace{\sum_{K=1}^{d} \sigma_K (u_K v_K^T)}_{\text{this is a rank-}d\text{ matrix.}} = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_d \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix} \begin{bmatrix} - & v_1^T & - \\ & \vdots & \\ - & v_d^T & - \end{bmatrix}$$

is $X^{(d)}$ a good rank-d approx to X?

- Question: does the greedy approach maximize var in all components? minimize error?

- **Question:** does the greedy approach maximize var in all components? minimize error? → is $X^{(d)} = Q^{(d)} Y^{(d)}$ an accurate approx. to X?

- yes. In fact, it is optimal...

  - recall: given $X = U\Sigma V^T$ with rank r, can write X via a sum of outer products

$$X = U\Sigma V^T = \begin{array}{c} \longleftarrow m \longrightarrow \\ \left[\begin{array}{cccc} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{array}\right] \\ \text{left singular vector} \end{array} \begin{array}{c} \text{singular value} \\ \longleftarrow n \longrightarrow \\ \left[\begin{array}{ccc|c} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \sigma_r & \\ \hline & & & \end{array}\right] \\ \leftarrow \text{rank}(A) \rightarrow \end{array} \begin{array}{c} \longleftarrow n \longrightarrow \\ \left[\begin{array}{c} - v_1^T - \\ - v_2^T - \\ \vdots \\ - v_n^T - \end{array}\right] \end{array} = \left[\begin{array}{cccc} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{array}\right] \left[\begin{array}{c} - \sigma_1 v_1^T - \\ \vdots \\ - \sigma_r v_r^T - \\ 0 \\ \vdots \\ 0 \end{array}\right] = \underbrace{\sum_{k=1}^{\text{rank}(A)} \sigma_k (u_k v_k^T)}_{\text{lin. comb. of outer products}}$$

  that is: $X = \sum_{k=1}^{\text{rank}(A)} \sigma_k (u_k v_k^T)$ is a sum of rank-one matrices $(u_k v_k^T)$ weighted by the corresponding singular value, σ_k

- earlier we saw: $X^{(d)} = (Q^{(d)} Q^{(d)})^T X = P_{\parallel Q} X = X_{\parallel Q^{(d)}}$

- if $Q^{(d)} = \left[\begin{array}{cccc} | & | & & | \\ u_1 & u_2 & \cdots & u_d \\ | & | & & | \end{array}\right]$ then $X^{(d)} = \underbrace{P_{\parallel \text{span}\{u_1, \dots, u_d\}} U}\Sigma V^T$

$$P_{\parallel \text{span}\{u_1,\dots,u_d\}} \left[\begin{array}{cccc|ccc} | & | & & | & | & & | \\ u_1 & u_2 & \cdots & u_d & u_{d+1} & \cdots & u_n \\ | & | & & | & | & & | \end{array}\right] = \left[\begin{array}{cccc|ccc} | & | & & | & | & & | \\ u_1 & u_2 & \cdots & u_d & 0 & \cdots & 0 \\ | & | & & | & | & & | \end{array}\right]$$

in span | $\perp$ to span | in span | $\perp$ to span

so, $X^{(d)} = P_{\parallel \{u_1, \dots, u_d\}} \sum_{k=1}^{r} \sigma_k (u_k v_k^T) = \sum_{k=1}^{d} \sigma_k (u_k v_k^T) + \sum_{k=d+1}^{r} 0 = \underbrace{\sum_{k=1}^{d} \sigma_k (u_k v_k^T)}_{\text{truncate sum at } d \text{ terms}}$

- truncate the sum at d terms... $X \approx X^{(d)} = \sum_{k=1}^{d} \sigma_k (u_k v_k^T)$
  - since $\sigma_1 \geq \sigma_2 \geq \sigma_3 \dots$ $X^{(d)} \approx X$ accurately for large enough d.

- Thm: (Eckart-Minsky-Young) given A, $m \times n$ w/ SVD $A = U\Sigma V^T$, and $\sigma_i \neq \sigma_j$ for $i \neq j$ then for $d \leq \text{rank}(A)$:

$$A^{(d)} = \sum_{j=1}^{d} \sigma_j (u_j v_j^T)$$

is the unique minimizer of $\|A - B\|_{Fro}$ over all B rank-d.

$\therefore$ $X^{(d)}$ is closest approx to X of rank-d. Minimizes $\|E^{(d)}\|_F^2 = \|X - X^{(d)}\|_F^2$, Maximizes $\|X^{(d)}\|_F^2$

- **Therefore:** the sequence $\{X^{(d)}\}_{d=1}^{\mathrm{rank}(X)}$ is <u>low-rank optimal</u>

rank 1: $X^{(1)} = \sigma_1(u_1 v_1^T)$ = best possible rank-1 approx to X

rank 2: $X^{(2)} = \sigma_1(u_1 v_1^T) + \sigma_2(u_2 v_2^T)$ = best possible rank-2 approx to X

rank 3: $X^{(3)} = \sigma_1(u_1 v_1^T) + \sigma_2(u_2 v_2^T) + \sigma_3(u_3 v_3^T)$ = best possible rank-3 approx to X

$\vdots$

rank d: $X^{(d)} = \sum_{j=1}^{d} \sigma_j (u_j v_j^T)$ = best possible rank-d approx to X.

- **Questions:**

1. how accurate is $X^{(d)}$? (what is $\|E^{(d)}\|_{Fro}^2$?)

   see HW.

   - $\|E^{(d)}\|_{Fro}^2 = \|X - X^{(d)}\|_{Fro}^2 = \sum_{j=1}^{\mathrm{rank}(X)} \sigma_j^2 - \sum_{j=1}^{d} \sigma_j^2 = \sum_{j>d} \sigma_j^2$

   so, relative error: $\|E^{(d)}\|_{Fro} / \|X\|_{Fro} = \left(\sum_{j>d}^{\mathrm{rank}(X)} \sigma_j^2 \Big/ \sum_{j=1}^{\mathrm{rank}(X)} \sigma_j^2 \right)^{1/2} = \left(1 - \sum_{j=1}^{d} \sigma_j^2 \Big/ \sum_{j=1}^{r} \sigma_j^2 \right)^{1/2} = \left(1 - \frac{\|X_d\|_{Fro}^2}{\|X\|_{Fro}^2}\right)^{1/2}$

2. when can we choose $d \ll n$?

   (when X is numerically low rank...)

   - when $\exists$ $d \ll n$ s.t. $\sum_{j=1}^{d} \sigma_j^2 \simeq \sum_{j=1}^{\mathrm{rank}(X)} \sigma_j^2$ $\Rightarrow$ when σ_j decay quickly

- **Ex:** Novembre et al, "Genes Mirror Geography Within Europe," Nature, 2008

  - **data:** $\left.\begin{array}{l} m = 1{,}387 \text{ individuals} \\ n = 197{,}146 \text{ loci} \end{array}\right\}$ but, much of genetic code likely shared by ancestry...

  - suggests $d \ll n$ possible and variation in data meaningful
  - seek d dim. subspace retaining/explaining as much variance as possible...

- <u>Where we go next:</u>

- <u>Questions</u>: (Evaluating PCA)

1. Related Methods: Multi-Dimensional Scaling ... What if we preserve pairwise distances? angles?
   Low Rank Matrix Completion... What if we are missing data?

2. why are many data matrices numerically low-rank?
   (see Townsend & Udell, Why are big data matrices low rank
   SIAM J. MATH. DATA SCIENCE, 2019)
   ↑
   nice application of Johnson-Lindenstrauss!

3. does PCA extract interpretable features?
   (see application example to game theory & strategic analysis in Poker → yes!
   see "eigenfaces" example → no!
   ... discuss limits of Frobenius norm → L_p low rank approx
   ... lack of realistic constraints in feature vectors, allow alternating / recursive corrections
   working from overall average to details → non-negative matrix factorization)

4. what if data is not concentrated on a low-d subspace?
   can we use <u>non</u>-linear representations? how do we study topology of data?
   (discuss intrinsic vs. extrinsic geometry, PCA depends on
   extrinsic geometry → diffusion maps, spectral graph embedding, topological data analysis.)

Thursday - May 18th - Low Rank Approximation

- **Logistics:**
  - HW 6 and Project 2 posted
    due next Thursday and Friday

- **Goals:**
  - Eckart-Mirsky-Young
    - Proof
    - Generic significance for low rank approximation
    - Ex:
      - Genes Mirror Geography
    - Looking ahead... (limitations and extensions)
  - Numerics and Computations going forward

Eckart-Mirsky-Young Thm:

- Thm: (Eckart-Minsky-Young) given A, $m \times n$ w/ SVD $A = U\Sigma V^*$, and $\sigma_i \neq \sigma_j$ for $i \neq j$
  then for $d \leq \text{rank}(A)$:
  $$A^{(d)} = \sum_{j=1}^{d} \sigma_j (u_j v_j^*)$$
  is the unique minimizer of $\|A - B\|_{Fro}$ over all B rank-d.

- Moreover: $\|A\|_{Fro}^2 = \sum_{j=1}^{\text{rank}(A)} \sigma_j^2$
  $\|A^{(d)}\|_{Fro}^2 = \sum_{j=1}^{d} \sigma_j^2$
  $\Big\}\ \|A - A^{(d)}\|_{Fro}^2 = \sum_{j=d+1}^{\text{rank}(A)} \sigma_j^2$

  so, relative error:
  $$\|A - A^{(d)}\|_{Fro} / \|A\|_{Fro} = \sqrt{1 - \left(\sum_{j=1}^{d} \sigma_j^2\right) / \left(\sum_{j=1}^{\text{rank}(A)} \sigma_j^2\right)}$$

- so, $A^{(d)} \approx A$ for $d \ll m,n$ if $\sigma_j(A)$ decay quickly past some small d

- numerical rank = smallest d s.t. $\|A^{(d)} - A\|_{Fro} / \|A\|_{Fro} \leq \underset{\text{desired error tolerance}}{\varepsilon} \in [0,1]$

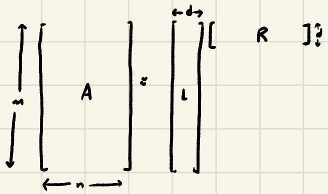
- **Fact:** many large matrices are numerically low rank... why? ⟵ (many large matrices are ill-conditioned)

  (see Townsend & Udell, Why are big data matrices low rank SIAM J. MATH. DATA SCIENCE, 2019)

  nice application of Johnson-Lindenstrauss!

  - so common often assumed w/out qualification
  - extremely useful, implies:

$$A^{(m\times n)} \approx \underset{\text{"left"}}{L^{(m\times d)}}\ \underset{\text{"right"}}{R^{(d\times n)}}$$



for $r \ll \min(m,n)$. Allows:

- Compression: store w/ $(m+n)d$ entries instead of mn
- Application: multiplying cost $\mathcal{O}((m+n)d)$ instead of $\mathcal{O}(mn)$
- Model reduction & low-dimensional embedding

- Let's prove Eckart-Mirsky-Young...

  - given $A \in \mathbb{C}^{m\times n}$, want $B \in \mathbb{C}^{m\times n}$ of rank $d \le \operatorname{rank}(A)$

    so write:

$$\underset{(m\times n)}{B} = \underset{(m\times d)}{L}\ \underset{(d\times n)}{R^*} = \sum_{k=1}^{d} \ell_k r_k^*$$

want $B \approx A$ so aim to minimize:

$$\|A-B\|_{Fro}^2 = \sum_{i,j=1}^{m,n} \Big| a_{ij} - \sum_{k=1}^{d}[\ell_k r_k^*]_{ij}\Big|^2 = \sum_{i,j=1}^{m,n}\Big| a_{ij} - \sum_{k=1}^{d} \ell_{ik}\bar{r}_{jk}\Big|^2$$

- two approaches:

  1. $\|\cdot\|_{Fro}$ is unitarily invariant since $\|M\|_{Fro}^2 = \overbrace{\operatorname{trace}(MM^*)}^{\langle M, M\rangle = \sum_{i,j} m_{ij}\cdot m_{ij}} = \sum_{j=1}^{\operatorname{rank}(M)} \sigma_j(M)^2$

     then $\|MQ^*\|_{Fro}^2 = \operatorname{trace}(MQ^*QM^*) = \operatorname{trace}(MM^*) = \|M\|_{Fro}^2$

     so, given $A = U\Sigma V^*$

$$\|A-B\|_{Fro}^2 = \|U^*(A-B)V\|_{Fro}^2 = \|\Sigma - (U^*L)(V^*R)^*\|_{Fro}^2 = \|\Sigma - \overbrace{\tilde{L}\tilde{R}^*}^{\tilde{B}}\|_{Fro}^2$$

where $\tilde{L} = U^*L \Leftrightarrow L = U\tilde{L}$, $\tilde{R} = V^*R \Leftrightarrow R = V\tilde{R}$

- WLOG can assume $A \in \mathbb{R}_+^{m\times n}$ and diagonal...

- if A is real and diagonal, $m \times n$ w/ nonnegative entries

then

$$A = \begin{bmatrix} \diagdown \\ \hdashline 0 \end{bmatrix} \quad \text{or} \quad \left[\begin{array}{c:c} \diagdown & 0 \end{array}\right]$$

in either case, only need to recover the nonzero block ($\text{rank}(A) \times \text{rank}(A)$) since any rows $= 0$ can be recovered by setting $\ell_{ik} = 0 \ \forall i$ s.t. row $i = 0$ and any col's $= 0$ ... $r_{jk} = 0 \ \forall j$ s.t. col. $j = 0$

- WLOG assume A is square \$ full rank
  real, nonnegative \$ diagonal } thus, square, Hermitian, positive definite

  (the rest is 0 blocks and unitary transformations by U and V)

- if A is symmetric then B best approximates A
  B^T best approximates A^T
  but $A^T = A$
  so $B^T = B$ (if unique)

look for symmetric, real solns:

$$L = R, \quad A \approx B = R R^T$$

$\uparrow$ real since A real

so, WLOG solve for $R \in \mathbb{R}^{r \times d}$ minimizing: $\underbrace{\| A - RR^T \|_{Fro}^2}_{f(R)} = \langle A - RR^T, A - RR^T \rangle = \sum_{i,j} (a_{ij} - \sum_{k=1}^{d} r_{ik} r_{jk})^2 = f(R)$

- reduced to a real, square, symmetric, p.d. problem...

  - symmetrizing solves 2 problems...

    (a) <u>scaling</u>: let D be a diagonal $d \times d$, invertible matrix...

    if $B = L R^T$

    then $B = L D D^{-1} R^T$ so (LD) and $(D^{-1} R^T)$ are also soln's $\forall D$

    (scale columns ℓ_k by d_{kk} and r_k by $1/d_{kk}$)

    - usually solve by either fixing $\|\ell_k\| = 1$, $\|r_k\| = 1$ or (symmetrized) $\|\ell_k\| = \|r_k\|$.

(b) orthogonality: if $B = LR^*$

can orthogonalize w/ QR decomp

$$L = \begin{bmatrix} \perp \text{ basis for } L \end{bmatrix} \begin{bmatrix} \diagdown \end{bmatrix} = Q_L T_L$$

or $R = Q_R T_R$

then:

$$B = Q_L \left[T_L T_R^* \right] Q_R^* = \underbrace{\begin{bmatrix} (n \times r) \\ \perp \text{ col's} \end{bmatrix}}_{\tilde{L}} \underbrace{\begin{bmatrix} r \times r \\ \text{invertible} \end{bmatrix} \begin{bmatrix} (r \times n) \\ \perp \text{ row's} \end{bmatrix}}_{\tilde{R}^*}$$

or $\underbrace{\qquad\qquad}_{\tilde{L}} \underbrace{\qquad\qquad}_{\tilde{R}^*}$

WLOG could require $\perp$ col's in L or R... which to enforce?

(once symmetrized we'll have both sides $\perp$...)

2. solve directly by computing $\nabla_R f(R)$ and setting $\nabla_R f(R) = 0$.

$$f(R) = \| A - RR^T \|_{Fro}^2 = \sum_{ij} \left(a_{ij} - (RR^T)_{ij} \right)^2 = \langle A - RR^T, A - RR^T \rangle$$ ← matrix (element-wise) inner product

$$\partial_{r_{\ell k}} f(R) = \partial_{r_{\ell k}} \langle A - RR^T, A - RR^T \rangle = \langle -\partial_{r_{\ell k}} RR^T, A - RR^T \rangle + \langle A - RR^T, -\partial_{r_{\ell k}} RR^T \rangle$$

$$= -2 \langle A - RR^T, \partial_{r_{\ell k}} RR^T \rangle$$

$$= -2 \sum_{ij} \left(a_{ij} - [RR^T]_{ij} \right) \left(\partial_{r_{\ell k}} [RR^T]_{ij} \right)$$

then: $[RR^T]_{ij} = \sum_{h=1}^{d} r_{ih} r_{jh}$ so $\partial_{r_{\ell k}} [RR^T]_{ij} = \sum_{h=1}^{d} \partial_{r_{\ell k}} r_{ih} r_{jh} = \partial_{r_{\ell k}} r_{ik} r_{jk} = \begin{cases} r_{ik} & \text{if } j = \ell \\ r_{jk} & \text{if } i = \ell \\ 2 r_{\ell k} & \text{if } i = j = \ell \\ 0 & \text{else} \end{cases}$

↑ $= 0$ unless $\left. \begin{array}{l} ih = \ell k \\ \text{or } jh = \ell k \end{array} \right\}$ $\ell = i$ or j, $k = h$

$$\text{so} \quad \partial_{r_{\ell k}} [RR^T] = \begin{array}{c} \xleftarrow{\quad n \quad} \\ \begin{bmatrix} 0 & r_{1k} & 0 \\ & r_{2k} & \\ & \vdots & \\ r_{1k} \cdots & 2 r_{\ell k} & r_{\ell+1,k} \cdots r_{nk} \\ & r_{\ell+1,k} & \\ 0 & \vdots & 0 \\ & r_{nk} & \end{bmatrix} \\ \xrightarrow{j} \ell = j \end{array} \begin{array}{l} \downarrow i \\ \ell = i \end{array} = r_k - e_\ell^T - + e_\ell - r_k^T -$$

k^{th} column of r

ℓ^{th} indicator

$$\text{so} \quad \partial_{r_{\ell k}} f(R) = -2 \sum_{i,j=1}^{n} (A - RR^T)_{ij} \left(r_k e_\ell^T + e_\ell r_k^T \right)_{ij}$$

for $\nabla_R f(R) = 0$ need $\partial_{r_{\ell k}} f(R) = 0 \;\; \forall \, \ell, k$

} can solve one k (col. of R at a time)

• So, to simplify, take greedy approach, solve one column of R at a time...

$$\partial_{r_{\ell k}} f(R) = -2 \sum_{ij} (A - RR^T)_{ij} \left(r_k e_\ell^T + e_\ell r_k^T \right)_{ij} = -2 \Big[\sum_{ij} (A - RR^T)_{ij} \underbrace{(r_k e_\ell^T)_{ij}}_{=0 \text{ unless } j=\ell} + \sum_{ij} (A - RR^T)_{ij} \underbrace{(e_\ell r_k)^T_{ij}}_{=0 \text{ unless } i=\ell} \Big]$$

$$= -2 \Big[\sum_i \underbrace{(A - RR^T)_{i\ell}}_{=(A-RR^T)_{\ell i} \text{ by symmetry}} r_{ik} + \sum_j (A - RR^T)_{\ell j} r_{jk} \Big]$$

$$= -4 \sum_j (A - RR^T)_{\ell j} r_{jk} = -4 \left[(A - RR^T) r_k \right]_\ell$$

so: $\nabla_{r_k} f(R) = -4 (A - RR^T) r_k \Rightarrow \nabla_{r_k} f(R) = 0$ requires $\underbrace{A r_k = RR^T r_k}_{\text{"generalized eigenvector problem"}}$ for all k

• if choose R w/ $\perp$ columns then: $R^T r_k = \begin{bmatrix} r_1^T r_k \\ r_2^T r_k \\ \vdots \\ r_k^T r_k \\ \vdots \\ r_d^T r_k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \|r_k\|^2 \\ \vdots \\ 0 \end{bmatrix}$

so $RR^T r_k = R(\|r_k\|^2 e_k) = \|r_k\|^2 r_k \ldots$

So, solving for R w/ $\perp$ col's that minimize $\|A - RR^T\|_{Fro}^2$ via gradient descent produces power iteration w/ shifts $\|r_k^{(\ell)}\|^2 \ldots$ (stage ℓ)

power iteration = grad. descent on low rank approx error!

so, using $\perp$ col's: $\nabla_{r_k} f(R) = -4(A r_k - \|r_k\|^2 r_k) = -4(A - \|r_k\|^2 I) r_k$

then: $\nabla_{r_k} f(R) = 0$ requires $A r_k = \|r_k\|^2 r_k$ ... eigenvector problem!

• <u>Conclusion</u>: if A is real, symmetric and $R \in \mathbb{R}^{m \times d}$ w/ $\perp$ col's minimizes $f(R) = \frac{1}{4} \|A - RR^T\|_{Fro}^2$

then each column r_k must be an eigenvector of A

where $A r_k = \lambda_k r_k$, and w/ magnitude $\|r_k\|^2 = \lambda_k$

so, if (v_j, λ_j) are the eigenpairs of A (normalized)

then, for all k, $\exists$ j s.t.

$$r_k = \sqrt{\lambda_j}\, v_j$$

• reduces the problem to assigning an eigenvector to each col of R...

• apparent that the best assignment is $r_k = \sqrt{\lambda_k}\, v_k$ where $|\lambda_1| \geq |\lambda_2| \geq \ldots$

• if $A = V \Lambda V^{-1} = V \Lambda V^T = \sum_{k=1}^{\text{rank}(A)} \lambda_k (v_k v_k^T)$

then, assigning $r_k = \sqrt{\lambda_k}\, v_k$ gives $RR^T = \sum_{k=1}^{d} (\sqrt{\lambda_k} v_k)(\sqrt{\lambda_k} v_k^T) = \sum_{k=1}^{d} \lambda_k (v_k v_k^T)$

so $A - RR^T = \sum_{k>d} \lambda_k (v_k v_k^T)$ and $\|A - RR^T\|_{Fro}^2 = \sum_{k>d} |\lambda_k|^2$

• to go back to generic A... $\text{rank}(A) = r$

$$\underset{m \times n}{\begin{bmatrix} A \end{bmatrix}} = \underset{m \times m}{\begin{bmatrix} U \end{bmatrix}} \begin{bmatrix} \Sigma \end{bmatrix} \underset{n \times n}{\begin{bmatrix} V^* \end{bmatrix}}$$

solve for R from the upper $r \times r$ block of Σ

$$\begin{bmatrix} \Sigma^{(1)} & 0 \\ 0 & 0 \end{bmatrix}, \quad \Sigma^{(1)} = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix}$$ has eigenpairs (σ_k, e_k)

so $r_k = \sqrt{\sigma_k}\, e_k$

• then, append zeros: $\tilde{l}_k = \sqrt{\sigma_k} \begin{bmatrix} e_k^{(1)} \\ 0 \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow m-r \end{matrix}$, $\tilde{r}_k = \sqrt{\sigma_k} \begin{bmatrix} e_k^{(1)} \\ 0 \end{bmatrix} \begin{matrix} \updownarrow r \\ \updownarrow n-r \end{matrix}$

then: $\tilde{L}\tilde{R}^T$ is the best rank d approximation to Σ

• and multiply by unitary transforms (change basis for row and column space)

$$l_k = U\tilde{l}_k = \begin{bmatrix} | & | & & | & & | \\ u_1 & u_2 & \cdots & u_k & \cdots & u_m \\ | & | & & | & & | \end{bmatrix} \sqrt{\sigma_k} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \sqrt{\sigma_k}\, u_k$$

$$r_k = V\tilde{r}_k = \begin{bmatrix} | & & | & & | \\ v_1 & \cdots & v_k & \cdots & v_n \\ | & & | & & | \end{bmatrix} \sqrt{\sigma_k} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} = \sqrt{\sigma_k}\, v_k$$

$$L = \text{diag}(\sigma_1, \dots \sigma_d)^{1/2} \begin{bmatrix} | & & | \\ u_1 & \cdots & u_d \\ | & & | \end{bmatrix}$$

$$R = \text{diag}(\sigma_1, \dots \sigma_d)^{1/2} \begin{bmatrix} | & & | \\ v_1 & \cdots & v_d \\ | & & | \end{bmatrix}$$

so $$B = L\,R^* = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_d \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{bmatrix} \begin{bmatrix} — & v_1^* & — \\ & \vdots & \\ — & v_d^* & — \end{bmatrix}$$

$$= \sum_{k=1}^{d} \sigma_k (u_k v_k^*) = A^{(d)}.$$

• Therefore, the best rank-d approximation to $A = A^{(d)} = \sum_{k=1}^{d} \sigma_k (u_k v_k^*)$
and is unique if $\sigma_1 > \sigma_2 > \dots$ $\blacksquare$

• <u>Consequence:</u> given $A \in \mathbb{C}^{m \times n}$, $M \in \mathbb{C}^{d \times d}$ invertible then the low rank decomp problem: find L, R given a normalization and $\perp$ constraint on one of the factors, $L \in \mathbb{C}^{m \times d}$, $R \in \mathbb{C}^{d \times n}$ minimizing $\|A - LMR^*\|_{Fro}^2$

can always be solved by truncating the SVD of A and is unique if σ_j are distinct. Allow accurate approx for $d \ll m, n$ if A is numerically low rank.

- Where we go next:

- Questions: (Evaluating PCA)

0. why are many data matrices numerically low-rank?
   or, when...

   (see Townsend & Udell, Why are big data matrices low rank
   SIAM J. MATH. DATA SCIENCE, 2019)

   nice application of Johnson-Lindenstrauss!

1. Related Methods: Multi-Dimensional Scaling ... What if we preserve pairwise distances? angles?
   Low Rank Matrix Completion... What if we are missing data?

2. does PCA extract interpretable features?
   (see application example to game theory & strategic analysis in Poker → yes!
   see "eigenfaces" example → no!
   ... discuss limits of Frobenius norm → Lp low rank approx
   ... lack of realistic constraints in feature vectors, allow alternating / recursive corrections
   working from overall average to details → non-negative matrix factorization)

3. what if data is not concentrated on a low-d subspace?
   can we use non-linear representations? how do we study topology of data?
   (discuss intrinsic vs. extrinsic geometry, PCA depends on
   extrinsic geometry → diffusion maps, spectral graph embedding, topological data analysis.)