Welcome to Ma221: Fall 2024, Lecture 2

Finish Lecture 1: 4 axes of NLA

1) math problems
2) structure
3) desired accuracy

   tradeoff: more accurate → slower

1) "Guaranteed accuracy": for $Ax=b$, need proof that A nonsingular
   ⇒ need arbitrary precision, very expensive
   → ask Mathematica
2) "Backward stable"
   informally: "get exact answer for a slightly wrong problem"
   Want $y = f(x)$
   get $alg(x) = f(x+\delta)$
   δ "small" compared to x
   def of "small" require matrix norms
   size of δ should be proportional to error in arithmetic, eg 10^{-16} = "machine epsilon" in double precision
   Then $alg(x)$ is "as good as data deserves" because computing x changes x to $x+\delta$

3) Residual as as small as desired:
for problems too big for a backward stable alg,

instead, use iterative algorithm to
progressively make residual smaller
Ex: make $\|Ax-b\|$ as small as desired
$\|Ax-b\|$ = backward error
lots of algorithms: Chap 6, 7

4) "Probably ok" → "randomized linear algebra"
Replace input, large, with a smaller one,
gotten by "randomly sampling" big one
then use deterministic method on small one
Some algs stop there, or iterate

Thm: the error is less than ε with
probability $1-\delta$ if size of random
approximation is large enough, ie.
size proportional to $f(\varepsilon,\delta)$, that gets
larger as ε, δ get smaller, typically

$$f(\delta,\varepsilon) = \Omega\left(\log\left(\frac{1}{\delta}\right)\cdot\frac{1}{\varepsilon^2}\right)$$

⇒ if your ε small need to iterate,
or use deterministic alg.

Some users want more info about reliability

1) Error bounds: note that if A singular, or "close", a backward stable alg can give completely wrong answer. We will derive error bounds proportional to

"condition number" = $\frac{1}{\text{distance from A to nearest singular matrix}}$

times machine epsilon, eg 10^{-16}

efficient to estimate $O(n^2)$ vs $O(n^3)$

2) "Guaranteed correct" except in "rare cases" (eg very close to singular): combine error bounds with a few steps of Newton, reasonable cost for $Ax=b$, LS

Popular because of 16-bit matrix multiply accelerators, eg Google TPUs

What about 8-bit floating point?

3) One more kind of "accuracy": getting bitwise identical results from run-to-run, eg to enable debugging.

Modern parallel computers may do sums in different order from run to run, floating point addition not associative

because of round-off ⇒ different sums

Axis 4: How to implement an "efficient algorithm"

Recall axis 2: counted flops - not only metric

1) Fewest keystrokes: eg "A\b" to solve $Ax=b$, Metric: minimizing human programming effort, will give pointers to available software, see class webpage (eg GAMS)
2) What does "fewest flops" mean?
   How many flops needed for n×n matmul?
   classical: $2n^3$
   Strassen (1969): $O(n^{\log_2 7}) = O(n^{2.81})$
   arbitrarily faster for large n, large hidden constant in $O(\,)$
   Coppersmith-Winograd (1987) $O(n^{2.376})$
   not practical, huge hidden constant
   Williams (2013) $O(n^{2.3728642})$
   Le Gall (2014) $O(n^{2.3728639})$
   Alman-Williams (2020) $O(n^{2.3728596})$
   Williams-Xu-Xu-Zhou (2024) $O(n^{2.371552})$
   D., Dumitriu, Holtz (2008) all other linear algebra problem can be solved

in $O(n^x)$ flops given matmul in $O(n^x)$ and backward stable, some practical ideas

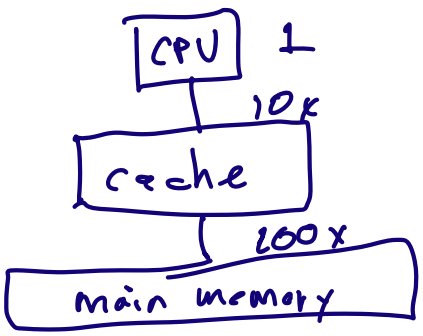
Will show how to reorganize $Ax=b$, LS to use fast matmul, even $O(n^3)$

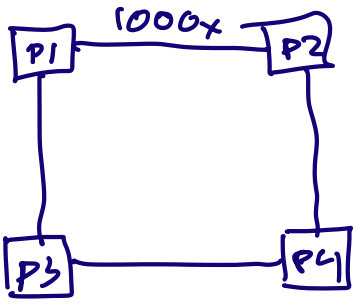
Conferences: BLIS Sep 26-27

3) Counting flops not only important metric

(3.1) Recall Moore's Law: observed #transistors on chip doubles every 2 years (until 2004). Now need to be parallel. Some parallel algs do same flops as sequential algs, other different, will discuss some of these algs, see CS267 too

(3.2) What is most expensive op?
Not arithmetic: moving data





Ex: "naive" 3 nested loops for matmul can be 100x slower than optimized version

lots of recent work to find "optimal" algs: minimize data movement

some optimal algs do same arithmetic just in diffent order, others need diffrnt algs

lots of links on webpage, possible projects

4) So far, we have minimized time
Energy also important (Moore's Law would have caused chips to melt, if continued)
Reasons to care about energy
battery in cell phone dying
how long your drone can fly
$1M / Megawatt / year to run data center
climate change
Turns out minimizing data movement also minimizes energy

Summary of Syllabus:
$Ax=b$, LS, eigenproblems, SVD, variations
Direct Methods (aka matrix factorizations)
LU, Cholesky, QR, Schur form, etc
LU: $A = P \cdot L \cdot U$
P = permutation
L = unit lower triangular
U = upper triangular

Cholesky: A sym, pos definite

$A = LL^T$ L = lower triangular

QR: $A = QR$ Q orthogonal (columns orthog)

R upper triagular

Eigenproblems (not Jordan Form)

Schur form (real case): $A = QRQ^T$

Q orthogonal

R upper triangular, evals on diagonal

(more complicated if complex evals)

SVD: $A = U\Sigma V^T$ U, V orthogonal

Σ diagonal

Iterative Algs:

Jacobi, Gauss-Seidel, Conjugate Gradient

Multigrid, etc.

Common idea: do many matrix-vector multiplies with A, find "best" linear combination of these vectors to approximate solution

Randomized Algs: Sampling A:

Approximate A by QA or QAV^T where Q and V are "skinny", always random, sometimes orthogonal, sparse, hybrids, then solve smaller problem with QA or QAV^T

+ structure
+ error bounds
+ algorithms for modern computers

Lecture 2:

Goals: Floating Point Arithmetic
Round off analysis
Beyond basics
exceptions: what is 0/0?

NaN = Not a Number

high/low/mixed precision
reproducibility
interval arithmetic

Ex: Polynomial evaluation and zero finding

Recall bisection: want to solve $f(x)=0$
start with $[x_1, x_2]$ $f(x_1)\cdot f(x_2)<0$
bisection: evaluate $f(\frac{x_1+x_2}{2})$
keep bisecting subinterval where f changes sign until narrow enough

Try $(x-2)(x-3)(x-4)$
$= x^3 - 9x^2 + 26x - 24$