CS 5220

Sparse linear algebra

David Bindel

2024-10-31

Goal

Solve

$$Ax = b,$$

where A is sparse (or data sparse).

Plan for today

- Reminder of stationary iterations
- Krylov idea and performance via
  - Parallelism in algorithm
  - Better convergence (preconditioning)
  - Better memory access (reordering)
- Sparse Gaussian elimination

Reminder: Stationary Iterations

From splitting $A = M - K$, compute:

$$x^{(k+1)} = x^{(k)} + M^{-1}(b - Ax^{(k)}).$$

"Linear" rate of convergence, dependent on $\rho(M^{-1}K)$.

Krylov Subspace Methods

What if we only know how to multiply by A? About all you can do is keep multiplying!

$$K_k(A, b) = \text{span}\left\{b, Ab, A^2b, \dots, A^{k-1}b\right\}.$$

Gives surprisingly useful information!

Example: Conjugate Gradients

If A is symmetric and positive definite, $Ax = b$ solves a minimization:

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b$$
$$\nabla\phi(x) = Ax - b.$$

Idea: Minimize $\phi(x)$ over $K_k(A, b)$. Basis for the *method of conjugate gradients*

Example: GMRES

Idea: Minimize $\|Ax - b\|^2$ over $K_k(A, b)$. Yields *Generalized Minimum RESidual* (GMRES)

Convergence of Krylov Subspace Methods

- KSPs are *not* stationary (no constant fixed-point iteration)
- Convergence is surprisingly subtle!
- CG convergence upper bound via *condition number*
  - Large condition number iff $\phi(x)$ is narrow
  - True for Poisson and company

Convergence of Krylov Subspace Methods

- *Preconditioned* problem $M^{-1}Ax = M^{-1}b$
- Whence M?
  - From a stationary method?
  - From a simpler/coarser discretization?
  - From approximate factorization?

PCG

```
r = b-A*x;
p = 0; beta = 0;
z = Msolve(r);
rho = dot(r, z);
for i=1:nsteps
    p = z + beta*p;
    q = A*p;
    alpha = rho/dot(p, q);
    x += alpha*p;
    r -= alpha*q;
    if norm(r) < tol, break; end
    z = Msolve(r);
    rho_prev = rho;
    rho = dot(r, z);
    beta = rho/rho_prev;
```

PCG parallel work

- Solve with M
- Product with A
- Dot products and axpys

Pushing PCG

- Rearrange if $M = LL^T$ is available
- Or build around “powers kernel”
  - Old “s-step” approach of Chronopoulos and Gear
  - CA-Krylov of Hoemmen, Carson, Demmel
  - Hard to keep stable

Pushing PCG

Two real application levers:

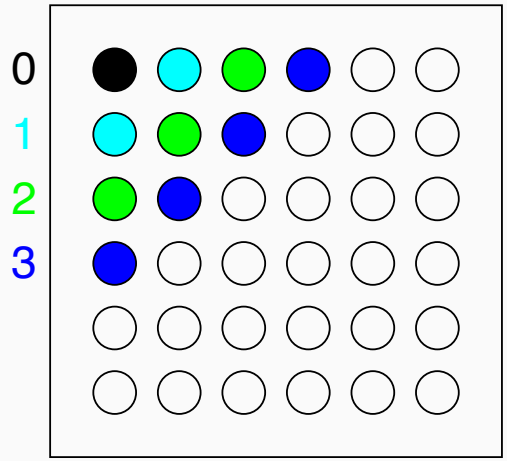
- Better preconditioning
- Faster matvecs

PCG bottlenecks

Key: fast solve with M, product with A

- Some preconditioners parallelize better!
- Balance speed with performance.
  - Speed for set up of M?
  - Speed to apply M after setup?
- Cheaper to do two multiplies/solves at once...
  - Can't exploit in obvious way — lose stability
  - Variants allow multiple products (CA-Krylov)
- Lots of fiddling possible with M; matvec with A?

Thinking on (basic) CG convergence



Consider 5-point stencil on an $n \times n$ mesh.

- Information moves one grid cell per matvec.
- Cost per matvec is $O(n^2)$.
- At least $O(n^3)$ work to get information across mesh!

Convergence by counting

- Time to converge $\geq$ time to move info across
- For a 2D mesh: $O(n)$ matvecs, $O(n^3) = O(N^{3/2})$ cost
- For a 3D mesh: $O(n)$ matvecs, $O(n^4) = O(N^{4/3})$ cost
- “Long” meshes yield slow convergence

Convergence by counting

3D beats 2D because everything is closer!

- Advice: sparse direct for 2D, CG for 3D.
- Better advice: use a preconditioner!

Eigenvalue approach

Define the *condition number* for $\kappa(L)$ s.p.d:

$$\kappa(L) = \frac{\lambda_{\max}(L)}{\lambda_{\min}(L)}$$

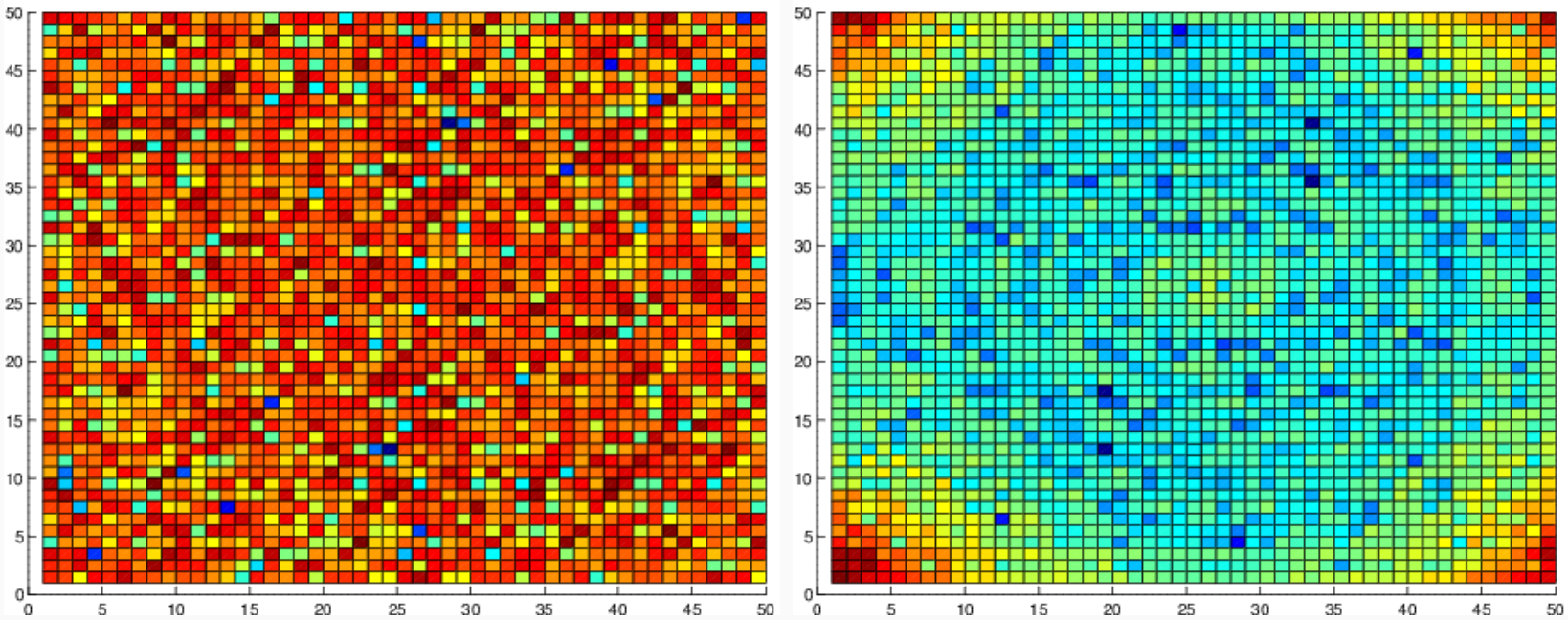
Describes how elongated the level surfaces of ϕ are.

Eigenvalue approach

- For Poisson, $\kappa(L) = O(h^{-2})$
- Steps to halve error: $O(\sqrt{\kappa}) = O(h^{-1})$.

Similar back-of-the-envelope estimates for some other PDEs. But these are not always that useful... can be pessimistic if there are only a few extreme eigenvalues.

Frequency-domain approach



Error e_k after k steps of CG gets smoother!

Preconditioning Poisson

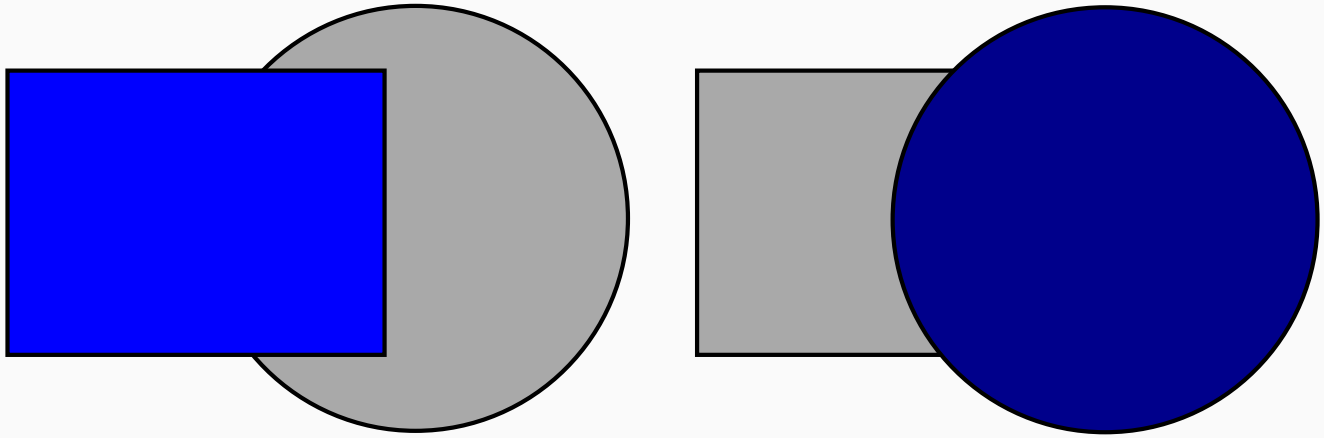
- CG already handles high-frequency error
- Want something to deal with lower frequency!
- Jacobi useless
    - Doesn’t even change Krylov subspace!

Preconditioning Poisson

Better idea: block Jacobi?

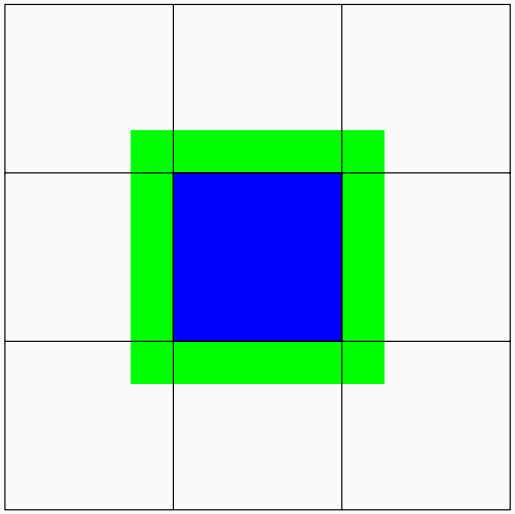
- Q: How should things split up?
- A: Minimize blocks across domain.
- Compatible with minimizing communication!

Multiplicative Schwartz



Generalizes block Gauss-Seidel

Restrictive Additive Schwartz (RAS)



- Get ghost cell data (green)
- Solve *everything* local (including neighbor data)
- Update local values for next step (local)
- Default strategy in PETSc

Multilevel Ideas

- RAS moves info one processor per step
- For scalability, still need to get around this!
- Basic idea: use multiple grids
    - Fine grid gives lots of work, kills high-freq error
    - Coarse grid cheaply gets info across mesh, kills low freq

Tuning matmul

Can also tune matrix multiply

- Represented implicitly (regular grids)
  - Example: Optimizing stencil operations (Datta)
- Or explicitly (e.g. compressed sparse column)
  - Sparse matrix blocking and reordering
  - Packages: Sparsity (Im), OSKI (Vuduc)
  - Available as PETSc extension

Or further rearrange algorithm (Hoemmen, Demmel).

Reminder: Compressed sparse row

```
for (int i = 0; i < n; ++i) {
  y[i] = 0;
  for (int jj = ptr[i]; jj < ptr[i+1]; ++jj)
    y[i] += A[jj]*x[col[jj]];
}
```

Where is the problem for memory access?

Memory traffic in CSR multiply

Memory access patterns:

- Elements of y accessed sequentially
- Elements of A accessed sequentially
- Access to x are all over!

Can help by switching to block CSR. Switching to single precision, short indices can help memory traffic, too!

Parallelizing matvec

- Each processor gets a piece
- Many partitioning strategies
- Idea: re-order so one of these strategies is “good”

Reordering for matvec

SpMV performance goals:

- Balance load?
- Balance storage?
- Minimize communication?
- Good cache re-use?

Reordering also comes up for GE!

Reminder: Sparsity and reordering

Permute unknowns for better SpMV or

- Stability of Gauss elimination,
- Fill reduction in Gaussian elimination,
- Improved performance of preconditioners...

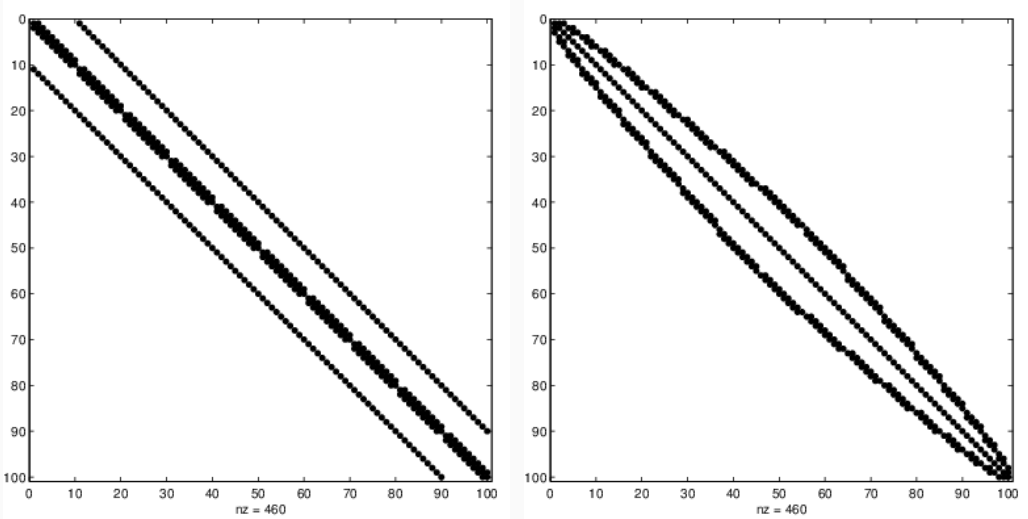
Reminder: Sparsity and partitioning

Want to partition sparse graphs so that

- Subgraphs are same size (load balance)
- Cut size is minimal (minimize communication)

Matrices that are “almost” diagonal are good?

Reordering for bandedness



Reverse Cuthill-McKee

- Select “peripheral” vertex v
- Order according to breadth first search from v
- Reverse ordering

From iterative to direct

- RCM ordering is great for SpMV
- But isn't narrow banding good for solvers, too?
  - LU takes $O(nb^2)$ where b is bandwidth.
  - Great if there's an ordering where b is small!

Skylines and profiles

- *Profile* solvers generalize band solvers
- Skyline storage for lower triangle: for each row i,
  - Start and end of storage for nonzeros in row.
  - *Contiguous* nonzero list up to main diagonal.
- In each column, first nonzero defines a profile.
- All fill-in confined to profile.
- RCM is again a good ordering.

Beyond bandedness

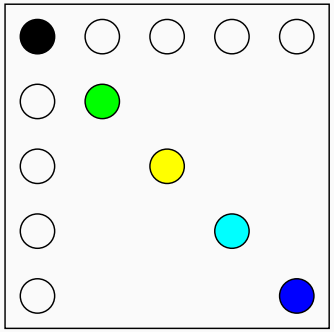
- Minimum bandwidth for 2D model problem? 3D?
- Skyline only gets us so much farther

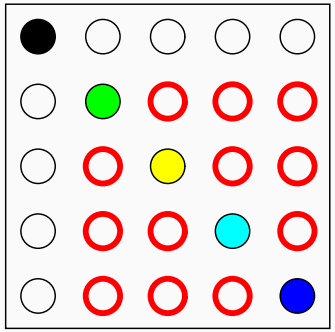
Beyond bandedness

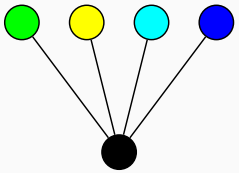
But more general solvers have similar structure

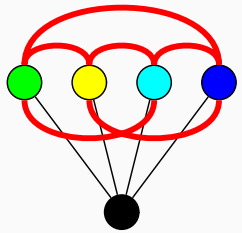
- Ordering (minimize fill)
- Symbolic factorization (where will fill be?)
- Numerical factorization (pivoting?)
- ... and triangular solves

Troublesome Trees



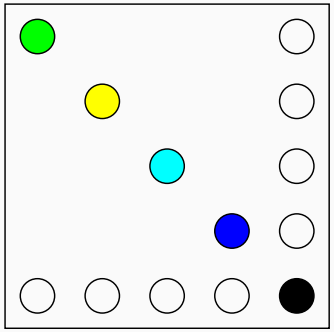


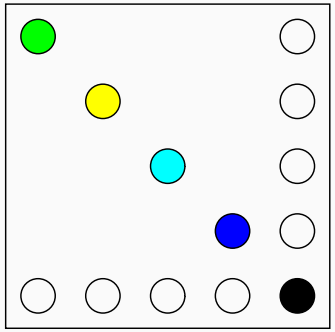


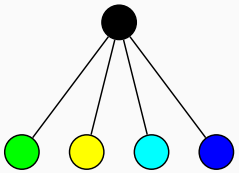


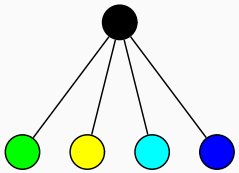
One step of Gaussian elimination *completely* fills this matrix!

Terrific Trees









Full Gaussian elimination generates *no* fill in this matrix!

Quiz

How many fill elements are there for elimination on

$$A = \begin{bmatrix} x & x & 0 & 0 & x & 0 & 0 & x \\ x & x & x & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & 0 \\ x & 0 & 0 & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & x & x & x & 0 \\ 0 & 0 & 0 & 0 & 0 & x & x & x \\ x & 0 & 0 & 0 & 0 & 0 & x & x \end{bmatrix}$$

Graphic Elimination

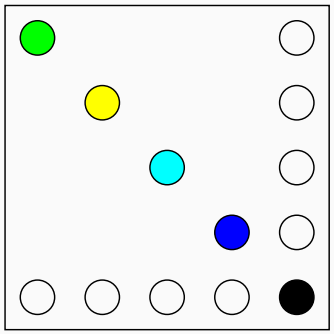
Consider first steps of GE

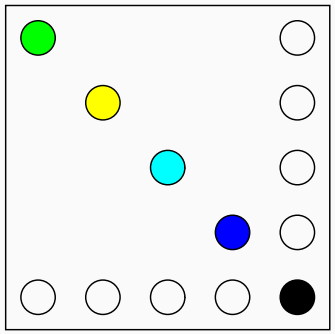
```
A(2:end,1)       = A(2:end,1)/A(1,1);
A(2:end,2:end) = A(2:end,2:end)-...
                     A(2:end,1)*A(1,2:end);
```

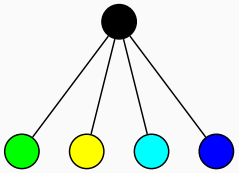
Nonzero in the outer product at (i, j) if A(i,1) and A(j,1) both nonzero — that is, if i and j are both connected to 1.

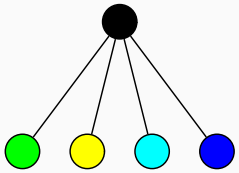
General: Eliminate variable, connect remaining neighbors.

Terrific Trees Redux





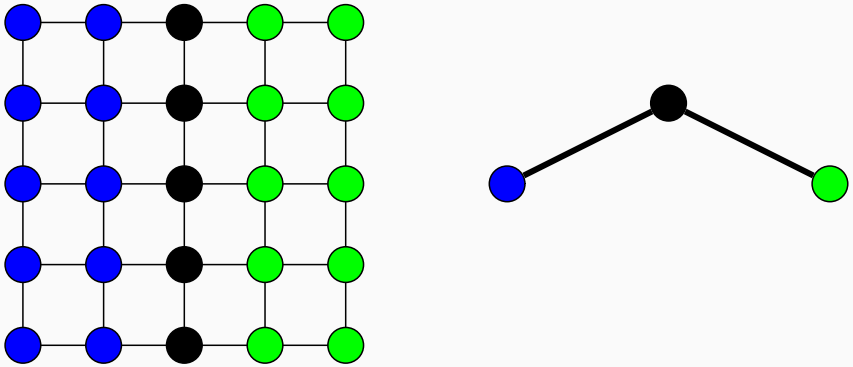




Order leaves to root $\Longrightarrow$

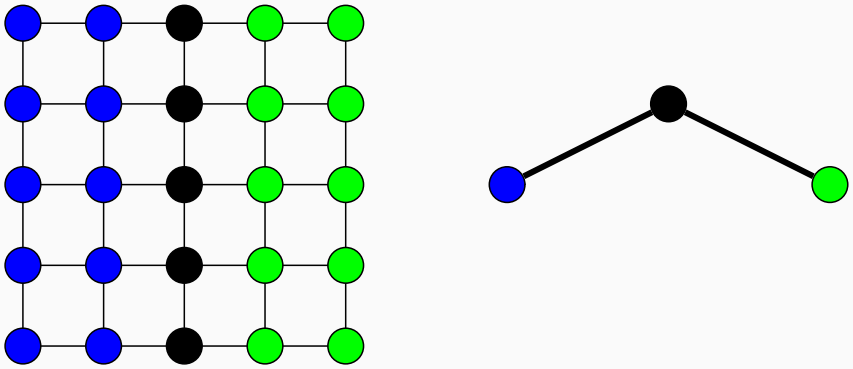
on eliminating i, parent of i is only remaining neighbor.

Nested Dissection



- Idea: Think of *block* tree structures.
- Eliminate block trees from bottom up.
- Can recursively partition at leaves.

Nested Dissection



- Rough cost estimate: how much just to factor dense Schur complements associated with separators?
- Notice graph partitioning appears again!
  - And again we want small separators!

Nested Dissection

Model problem: Laplacian with 5 point stencil (for 2D)

- ND gives optimal complexity in exact arithmetic (George 73, Hoffman/Martin/Rose)
- 2D: $O(N \log N)$ memory, $O(N^{3/2})$ flops
- 3D: $O(N^{4/3})$ memory, $O(N^2)$ flops

Minimum Degree

- Locally greedy strategy
  - Want to minimize upper bound on fill-in
  - Fill $\leq$ (degree in remaining graph)^2
- At each step
  - Eliminate vertex with smallest degree
  - Update degrees of neighbors
- Problem: Expensive to implement!
  - But better varients via *quotient graphs*
  - Variants often used in practice

Elimination Tree

- Variables (columns) are nodes in trees
- j a descendant of k if eliminating j updates k
- Can eliminate disjoint subtrees in parallel!

Cache locality

Basic idea: exploit “supernodal” (dense) structures in factor

- e.g. arising from elimination of separator Schur complements in ND
- Other alternatives exist (multifrontal solvers)

Pivoting

Pivoting is painful, particularly in distributed memory!

- Cholesky — no need to pivot!
- Threshold pivoting — pivot when things look dangerous
- Static pivoting — try to decide up front

What if things go wrong with threshold/static pivoting?

Common theme: Clean up sloppy solves with good residuals

Direct to iterative

Can improve solution by *iterative refinement*:

$$
\begin{aligned}
PAQ &\approx LU \\
x_0 &\approx QU^{-1}L^{-1}Pb \\
r_0 &= b - Ax_0 \\
x_1 &\approx x_0 + QU^{-1}L^{-1}Pr_0
\end{aligned}
$$

Looks like approximate Newton on $F(x) = Ax - b = 0$.

This is just a stationary iterative method!

Nonstationary methods work, too.

Variations on a theme

If we're willing to sacrifice some on factorization,

- Single precision factor + double precision refinement?
- Sloppy factorizations (marginal stability) + refinement?
- Modify m small pivots as they're encountered (low rank updates), fix with m steps of a Krylov solver?

Parting advice

- Sparse direct for 2D problems
- Gets more expensive for 3D problems
- Approximate direct solves make good preconditioners