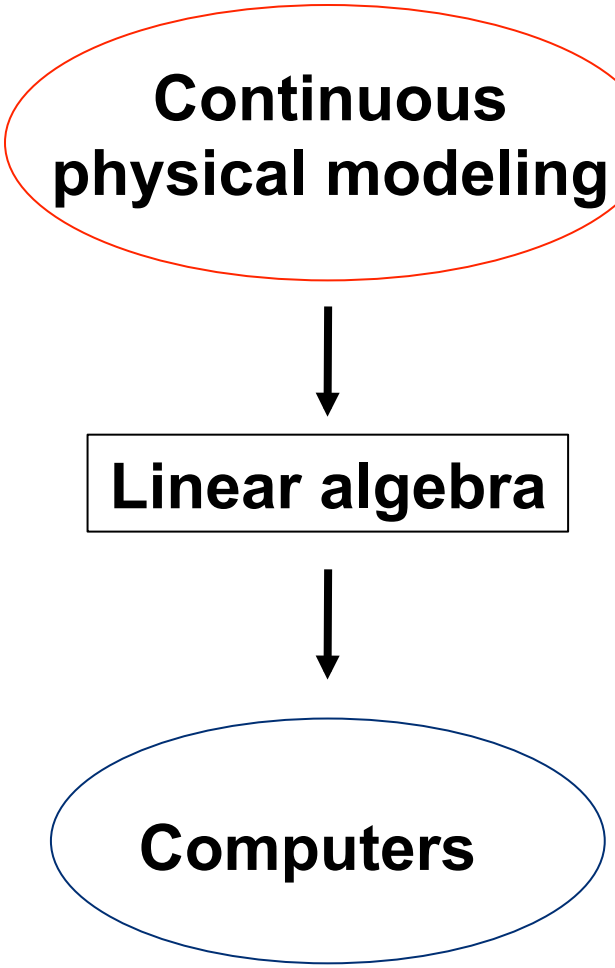
The middleware of scientific computing



$$Ax = b$$

Conjugate gradient iteration

- CG can be used to solve *any* system $Ax = b$, if …

Conjugate gradient iteration

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

Conjugate gradient iteration

- CG can be used to solve *any* system $Ax = b$, if …
- The matrix A is *symmetric* ($a_{ij} = a_{ji}$) …
- … and *positive definite* (all eigenvalues > 0).

- Symmetric positive definite matrices occur a lot in scientific computing & data analysis!

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$x_k = x_{k-1} + \ldots$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = \ldots$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$d_k = \ldots$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = \ldots$ new residual

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$

$d_k = r_k + \beta_k d_{k-1}$ new search direction

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ new approx solution

$r_k = r_{k-1} - \alpha_k A d_{k-1}$ new residual

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$

$d_k = r_k + \beta_k d_{k-1}$ new search direction

Conjugate gradient iteration to solve A*x=b

$x_0 = 0, \quad r_0 = b, \quad d_0 = r_0$ (these are all vectors)

for $k = 1, 2, 3, \ldots$

$\alpha_k = (r^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$ step length

$x_k = x_{k-1} + \alpha_k d_{k-1}$ approximate solution

$r_k = r_{k-1} - \alpha_k A d_{k-1}$ residual = $b - Ax_k$

$\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$ improvement

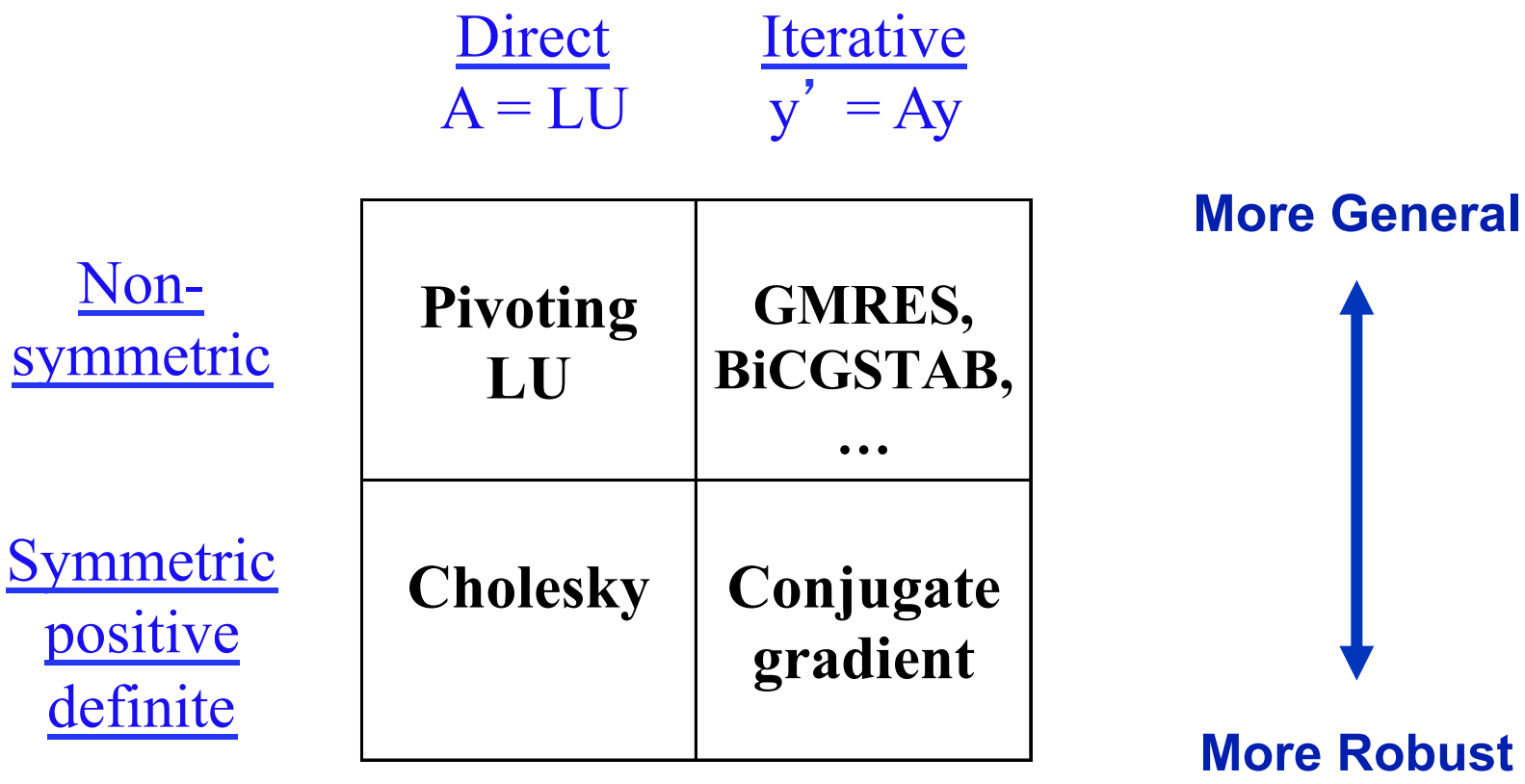
$d_k = r_k + \beta_k d_{k-1}$ search direction

- One matrix-vector multiplication per iteration
- Two vector dot products per iteration
- Four n-vectors of working storage

Vector and matrix primitives for CG

- DAXPY: $v = \alpha * v + \beta * w$ (vectors v, w; scalars α, β)
  - Time = O(n)
- DDOT: $\alpha = v^T * w = \sum_j v[j] * w[j]$ (vectors v, w; scalar α)
  - Time = O(n)
- Matvec: $v = A * w$ (matrix A, vectors v, w)
  - This is the hard part!
  - Time = $O(n^2)$ if A is a full matrix stored as a 2-D array
  - But all you need is a subroutine to compute v from w
  - If A is *sparse*, time = O(#nonzeros in A)

The Landscape of Ax=b Solvers



Optional:

Analysis of the Conjugate Gradient Algorithm

See Shewchuk's paper (linked to course web site) for details.

Conjugate gradient: Krylov subspaces

- Eigenvalues: $Av = \lambda v$ $\{ \lambda_1, \lambda_2, \ldots, \lambda_n \}$
- Cayley-Hamilton theorem:

$$(A - \lambda_1 I) \cdot (A - \lambda_2 I) \cdot \cdots \cdot (A - \lambda_n I) = 0$$

Therefore $\sum_{0 \le i \le n} c_i A^i = 0$ for some c_i

so $A^{-1} = \sum_{1 \le i \le n} (-c_i / c_0) A^{i-1}$

- Krylov subspace:

Therefore if $Ax = b$, then $x = A^{-1} b$ and

$x \in \text{span}(b, Ab, A^2 b, \ldots, A^{n-1} b) = K_n(A, b)$

Conjugate gradient: Orthogonal sequences

- Krylov subspace: $K_i(A, b) = \text{span}(b, Ab, A^2b, \ldots, A^{i-1}b)$
- Conjugate gradient algorithm:

  for i = 1, 2, 3, . . .

  find $x_i \in K_i(A, b)$

  such that $r_i = (b - Ax_i) \perp K_i(A, b)$

- Notice $r_i \in K_{i+1}(A, b)$, so $r_i \perp r_j$ for all $j < i$
- Similarly, the "directions" are A-orthogonal:

$$(x_i - x_{i-1})^T \cdot A \cdot (x_j - x_{j-1}) = 0$$

- The magic: Short recurrences. . .

  A is symmetric => can get next residual and direction from the previous one, without saving them all.

Conjugate gradient: Convergence

- In exact arithmetic, CG converges in n steps
  (completely unrealistic!!)
- Accuracy after k steps of CG is related to:
  - consider polynomials of degree k that are equal to 1 at 0.
  - how small can such a polynomial be at all the eigenvalues of A?
- Thus, eigenvalues close together are good.
- Condition number: $\kappa(A) = \|A\|_2 \|A^{-1}\|_2 = \lambda_{max}(A) / \lambda_{min}(A)$
- Residual is reduced by a constant factor by
  $O(\kappa^{1/2}(A))$ iterations of CG.

Other Krylov subspace methods

- Nonsymmetric linear systems:
  - GMRES:

    for i = 1, 2, 3, . . .

    find $x_i \in K_i(A, b)$ such that $r_i = (Ax_i - b) \perp K_i(A, b)$

    But, no short recurrence => save old vectors => lots more space

    (Usually "restarted" every k iterations to use less space.)

  - BiCGStab, QMR, etc.:

    Two spaces $K_i(A, b)$ and $K_i(A^T, b)$ w/ mutually orthogonal bases

    Short recurrences => O(n) space, but less robust

  - Convergence and preconditioning more delicate than CG
  - Active area of current research
- Eigenvalues: Lanczos (symmetric), Arnoldi (nonsymmetric)