Iterative Solvers of Ax = b
The Conjugate Gradient Method

CS 111: Introduction to Computational Science

Spring 2019 Lecture #8

Ziad Matni, Ph.D.

Administrative

- Homework #4 due next Tuesday

- Midterm #1 is NEXT week Thursday (May 2nd)
  - Review questions are up!
  - Discuss further in section today

Lecture Outline

- Iteration to solve Ax = b:
  Conjugate Gradient Method

Ax = b Solvers We've Learned So Far

- Pivoting
- LU Factorization
- Cholesky
- QR Factorization
- Jacobi's Method

Conjugate Gradient Iteration

- Another (more efficient) iterative algorithm to solve $Ax = b$.
- Start with a guess $x^{(0)}$, then compute $x^{(1)}$, $x^{(2)}$, ...
- Stop when you think you're close enough.

- In theory, CG can be used to solve *any* system $Ax = b$, provided only that **A is SPD**.

- In practice, how well CG works depends on specifics of **A** in subtle ways, involving eigenvalues and condition number.

Conjugate Gradient Iteration for **Ax = b**

$x^{(0)} = 0$	approximate solution
$r^{(0)} = b$	residual = b - Ax
$d^{(0)} = r^{(0)}$	search direction
for $k = 1, 2, 3, \ldots$	
$\quad \alpha^{(k)} = (r^{(k-1)\mathbf{T}}\, r^{(k-1)}) \,/\, (d^{(k-1)\mathbf{T}} A\, d^{(k-1)})$	step length
$\quad x^{(k)} = x^{(k-1)} + \alpha^{(k)}\, d^{(k-1)}$	new approx. solution
$\quad r^{(k)} = r^{(k-1)} - \alpha^{(k)} A\, d^{(k-1)}$	new residual
$\quad \beta^{(k)} = (r^{(k)\mathbf{T}}\, r^{(k)}) \,/\, (r^{(k-1)\mathbf{T}}\, r^{(k-1)})$	improvement
$\quad d^{(k)} = r^{(k)} + \beta^{(k)}\, d^{(k-1)}$	new search direction

Conjugate Gradient Iteration for **Ax = 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: $\alpha = v^T * w = \Sigma_j v[j] * w[j]$ (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

	Direct $A = LU$	Iterative $y^{(k+1)} = Ay^{(k)}$
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
SPD	**Cholesky**	**Conjugate gradient**

More General $\updownarrow$ **More Specific**

More Robust $\leftrightarrow$ **Less Storage (if sparse)**

Examples!

- See blackboard and Jupyter Notebook

Your To-Dos

- Homework #4 – due **Monday, April 29th**
- **Study for your midterm next week!** ☺

</LECTURE>