CONVEX OPTIMIZATION PROBLEMS

MAT 220

Dr.C. Rajan (8113053359) Off: S109E

The Landscape of Ax=b Solvers

	Direct A = LU	Iterative y' = Ay
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite	**Cholesky**	**Conjugate gradient**

More General

More Robust

More Robust ⟷ **Less Storage (if sparse)**

D

METHODS TO SOLVE LINEAR SYSTEMS

- Direct methods

  Gaussian elimination method

  LU method for factorization

  Simplex method of linear programming
- Iterative method

  Jacobi method

  Gauss-Seidel method

  Multi-grid method

  Conjugate gradient method

Conjugate Gradient method

- The CG is an algorithm for the numerical solution of particular system of linear equations Ax=b.

  Where A is symmetric i.e., $A = A^T$ and positive definite i.e.,

$$x^T * A * x > 0 \text{ for all nonzero vectors}$$

If A is symmetric and positive definite then the function

$$Q(x) = \frac{1}{2} x`Ax - x`b + c$$

Conjugate gradient method

- Conjugate gradient method builds up a solution $x^* \in R^n$ in at most n steps in the absence of round off errors.
- Considering round off errors more than n steps may be needed to get a good approximation of exact solution x^*
- For sparse matrices a good approximation of exact solution can be achieved in less than n steps in also with round off errors.

Practical Example

In oil reservoir simulation,

The number of linear equations corresponds to the number of grids of a reservoir

- The unknown vector x is the oil pressure of reservoir
- Each element of the vector x is the oil pressure of a specific grid of the reservoir

Linear System

$$Ax = b$$

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Square matrix

Unknown vector
(what we want to find)

Known vector

Matrix Multiplication

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \boxed{b_1} \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Positive Definite Matrix

A is *positive-definite* if, for every nonzero vector x,

$$x^T A x > 0.$$

$$[\, x_1 \;\; x_2 \; \ldots \; x_n \,] \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} > 0$$

Procedure

- Finding the initial guess for solution x_0
- Generates successive approximations to x_0
- Generates residuals
- Searching directions

Conjugate gradient iteration

- $x_0 = 0, \quad r_0 = b, \quad p_0 = r_0$
- **<u>for</u>** $k = 1, 2, 3, \ldots$
- $\alpha_k = (r^T_{k-1} r_{k-1}) / (p^T_{k-1} A p_{k-1})$ step length
- $x_k = x_{k-1} + \alpha_k p_{k-1}$ approximate solution
- $r_k = r_{k-1} - \alpha_k A p_{k-1}$ residual
- $\beta_k = (r^T_k r_k) / (r^T_{k-1} r_{k-1})$ improvement
- $p_k = r_k + \beta_k p_{k-1}$ search direction

- Iteration of conjugate gradient method is of the form

$$x(t) = x(t-1) + s(t)d(t)$$

where,

$x(t)$ is function of old value of vector x

$s(t)$ is scalar step size x

$d(t)$ is direction vector

Before first iteration, values of $x(0)$, $d(0)$ and $g(0)$ must be set

Steps to find conjugate gradient method

- Every iteration t calculates x(t) in four steps :

Step 1: Compute the gradient

$$g(t) = Ax(t-1) - b$$

Step 2: Compute direction vector

$$d(t) = - g(t) + [g(t)` g(t) / g(t-1)` g(t-1)] d(t-1)$$

Step 3: Compute step size

$$s(t) = [- d(t)` g(t)]/d(t)' A d(t)]$$

Step 4: Compute new approximation of x

$$x(t) = x(t-1) + s(t)d(t)$$

Sequential Algorithm

1) $x0 = 0$

2) $r0 := b - Ax0$

3) $p0 := r0$

4) $k := 0$

5) $Kmax :=$ *maximum number of iterations* to be done

6) if $k < kmax$ *then perform 8 to 16*

7) if $k = kmax$ *then exit*

8) calculate $v = Ap_k$

9) $\alpha_k := r_k^T r_k / p^T_k v$

10) $x_{k+1} := x_k + a_k p_k$

11) $r_{k+1} := r_k - a_k v$

12) if r_{k+1} *is sufficiently small then go to 16*

end if

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

14) $p_{k+1} := r_{k+1} + \beta_k p_k$

15) $k := k + 1$

16) $result = x_k + 1$

Complexity analysis

- To Identify Data Dependencies
- To identify eventual communications
- Requires large number of operations
- As number of equations increases complexity also increases .

Conjugate gradient iteration for Ax = b

$x_0 = 0$ approx solution

$r_0 = b$ residual = b - Ax

$d_0 = r_0$ search direction

<u>for</u> $k = 1, 2, 3, \ldots$

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

<u>for</u> $k = 1, 2, 3, \ldots$

$\alpha_k = \ldots$ 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

<u>for</u> $k = 1, 2, 3, \ldots$

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

<u>for</u> $k = 1, 2, 3, \ldots$

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

<u>for</u> $k = 1, 2, 3, \ldots$

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

$x_0 = 0, \quad r_0 = b, \quad d_0 = r_0$

<u>for</u> $k = 1, 2, 3, \ldots$

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

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

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

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

Conjugate gradient: Krylov subspaces

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

- Cayley-Hamilton theorem:

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

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:

  <u>for</u> $i = 1, 2, 3, \ldots$

  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:

    <u>for</u> $i = 1, 2, 3, \ldots$

    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)

Preconditioners

- Suppose you had a matrix B such that:

  1. condition number $\kappa(B^{-1}A)$ is small
  2. $By = z$ is easy to solve

- Then you could solve $(B^{-1}A)x = B^{-1}b$ instead of $Ax = b$

- B = A is great for (1), not for (2)
- B = I is great for (2), not for (1)
- Domain-specific approximations sometimes work
- B = diagonal of A sometimes works

- Better: blend in some direct-methods ideas. . .

Preconditioned conjugate gradient iteration

$x_0 = 0, \quad r_0 = b, \quad d_0 = B^{-1} r_0, \quad y_0 = B^{-1} r_0$

<u>for</u> $k = 1, 2, 3, \ldots$

$\alpha_k = (y^T_{k-1} r_{k-1}) / (d^T_{k-1} A d_{k-1})$	step length
$x_k = x_{k-1} + \alpha_k d_{k-1}$	approx solution
$r_k = r_{k-1} - \alpha_k A d_{k-1}$	residual
$y_k = B^{-1} r_k$	preconditioning solve
$\beta_k = (y^T_k r_k) / (y^T_{k-1} r_{k-1})$	improvement
$d_k = y_k + \beta_k d_{k-1}$	search direction

- One matrix-vector multiplication per iteration
- One solve with preconditioner per iteration

The Landscape of Ax=b Solvers

	Direct A = LU	Iterative y' = Ay
Non-symmetric	**Pivoting LU**	**GMRES, BiCGSTAB, …**
Symmetric positive definite	**Cholesky**	**Conjugate gradient**

More General

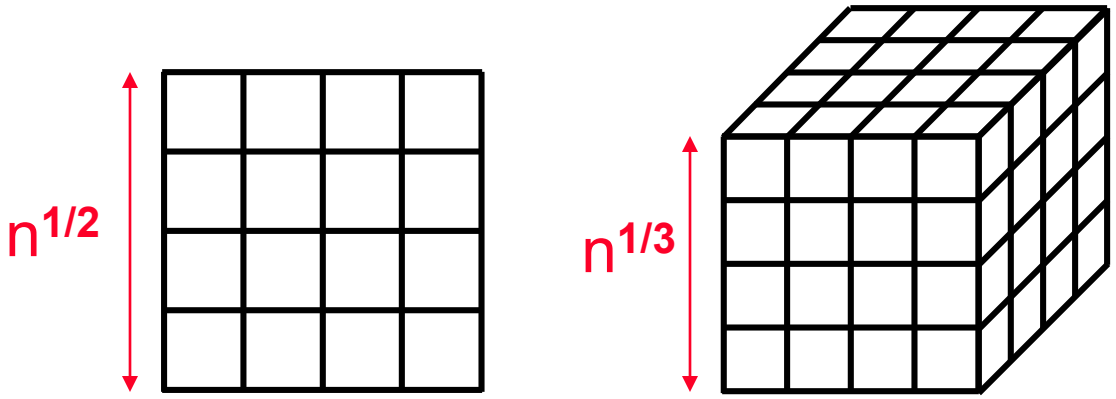
More Robust

More Robust ⟷ **Less Storage (if sparse)**

D

Complexity of linear solvers

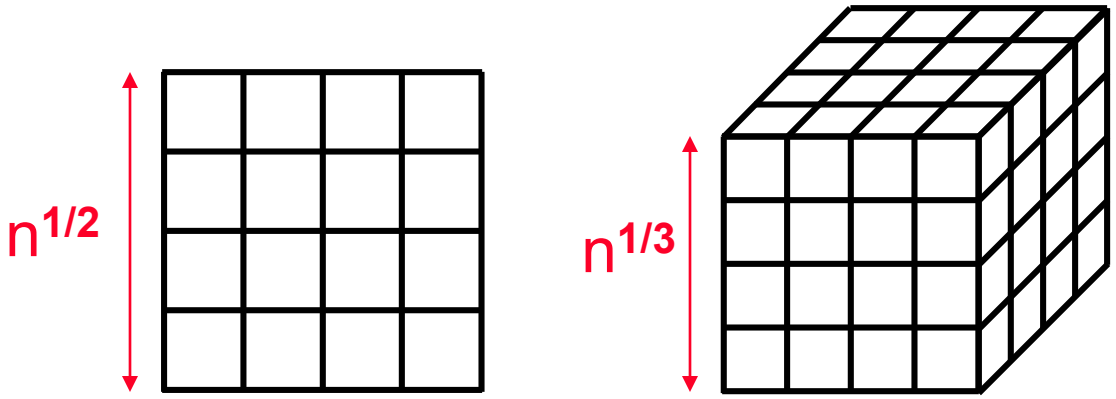
Time to solve model problem (Poisson's equation) on regular mesh



	2D	3D
Sparse Cholesky:	$O(n^{1.5})$	$O(n^{2})$
CG, exact arithmetic:	$O(n^{2})$	$O(n^{2})$
CG, no precond:	$O(n^{1.5})$	$O(n^{1.33})$
CG, modified IC:	$O(n^{1.25})$	$O(n^{1.17})$
CG, support trees:	$O(n^{1.20})$ -> $O(n^{1+})$	$O(n^{1.75})$ -> $O(n^{1.31})$
Multigrid:	$O(n)$	$O(n)$

Complexity of direct methods

Time and space to solve any problem on any well-shaped finite element mesh



	2D	3D
Space (fill):	$O(n \log n)$	$O(n^{4/3})$
Time (flops):	$O(n^{3/2})$	$O(n^{2})$