

Lecture 14: CMSC 878R/AMSC698R

Iterative Methods and the FMM

Outline

- Direct Solution of Linear Systems
 - Inverse, LU decomposition, Cholesky, SVD, etc.
- Iterative methods for linear systems
 - Why?
- Matrix splittings and fixed-point schemes
 - SOR, Jacobi, Gauss Seidel, etc.
- Krylov Methods
 - Conjugate Gradient, GMRES
- Convergence and Preconditioning
- Eigenvalue Problems
- Iterative methods for eigenvalue problems

Preliminary Comments

- FMM accelerates matrix-vector multiplication
- However, most problems require solution of linear systems
- Goal: See how linear systems can be solved using FMM
- Take home message
 - Iterative methods require matrix vector products
 - Can be written as FMM
 - Some methods can be guaranteed to converge in N steps
 - With good guess and clever algorithms may converge much faster
- Philosophy:
 - In general it is better to be a “consumer” of linear algebra
 - Use canned software which asks you to provide a product routine
 - However, with FMM there may not be “canned” or “tuned” software available and we may have to develop our own

Direct Solution of linear systems

$$Ax=b \rightarrow x=A^{-1}b$$

- Almost never a good idea to construct inverse to solve
- Computing inverse is more expensive than solving system
 - Computing inverse leads to loss of precision
- Exceptions
 - Analytical expression for inverse can be obtained, e.g.,

Sherman Morrison Woodbury

$$(A + u \otimes v)^{-1} = A^{-1} - \frac{(A^{-1}u) \otimes (v \cdot A^{-1})}{1 + \lambda}, \quad \lambda \equiv v \cdot A^{-1}u.$$

Woodbury Formula

$$(A + UV^T)^{-1} = A^{-1} - [A^{-1}U(1 + V^T A^{-1}U)^{-1}V^T A^{-1}].$$

Direct solutions (2)

- Consumer level – Use LAPACK or MATLAB,
 - (vectorized, cache-optimized, etc.) -- ATLAS
- For non symmetric matrices use LU decomposition
- LU decomposition $\simeq N^3/3$ operations
- Backsubstitution $O(N^2)$
- For symmetric matrices use Cholesky decomposition $\simeq N^3/6$ operations
- All these methods require storage of the matrix and for dense matrices have memory complexity $O(N^2)$

References

- C.T. Kelley, “Iterative methods for Linear and Nonlinear Equations, SIAM, 1995)
- J. Shewchuck, “An Introduction to the Conjugate Gradient Method Without the Agonizing Pain”
 - (downloadable from <http://www-2.cs.cmu.edu/~jrs/jrspapers.html>)
- “Templates for the solution of linear systems: Building Blocks and Iterative Methods,” Barrett et al, SIAM (downloadable at http://www.netlib.org/linalg/html_templates/Templates.html)
- Yousef Saad has two good books online
 - Numerical Methods for Large Eigenvalue Problems*
 - Iterative methods for sparse linear systems.* (downloadable at <http://www-users.cs.umn.edu/~saad/books.html>)

Iterative Methods: Notation

$$Ax = b$$

- A is a nonsingular $N \times N$ matrix, $x, b \in \mathbb{R}^N$
- Ax can be calculated using FMM
- x^* is the solution to be found. $x^* = A^{-1}b \in \mathbb{R}^N$
- Definitions

Norm of A	$\ A\ = \max_{\ x\ =1} \ Ax\ $
Condition number of A	$\kappa(A) = \ A\ \ A^{-1}\ $
Residual	$r = b - Ax$
Error	$e = x - x^*$

Fixed point iteration

- In fixed point iteration we write $x = Mx$
- If M is a contraction ($\|M\| < 1$) then following converges
 - Start with guess x_0
 - Generate successive estimates $x_k = M x_{k-1}$
- How to write our equation as a fixed point scheme?

$$Ax = b \Rightarrow (I - A)x = -b$$

- So, $I x = (I - A)x + b$
- (Richardson iteration) $x_{k+1} = (I - A)x_k + b$
- For convergence we require $\|I - A\| < 1$
- Iteration Matrix M here is $I - A$

Left Preconditioning

- What if $\|I-A\| = \text{or} > 1$?
- Let B be another matrix.
- Then $BAx=Ix + (BA-I)x = Bb$
- So, $I x = (I-BA)x + Bb$
- (Richardson iteration) $x_{k+1} = (I-BA)x_k + Bb$
- For convergence we require $\|I-BA\| < 1$
- If B is simple (e.g., diagonal) this is easy to compute
- Iteration matrix $M = \|I-BA\|$
- This is left preconditioning
- Can also device right preconditioning

Right Preconditioning

- Let B be another matrix.
- Let $AB^{-1}u=b$ and $u=Bx$
- Then $AB^{-1}(Bx)=b$
- So, $x_{k+1} = x_k - AB^{-1}u_k + b$
- Compute $u_k = Bx_k$

Classical fixed point methods

- Write $A=A_1+A_2$
- Then iteration becomes $x_{k+1} = A_1^{-1}(b - A_2x_k)$
 - For convergence $\|A_1^{-1}A_2\| < 1$
 - A_1^{-1} should be easy to compute
 - In addition the FMM should be used to compute A_2x_k
- Jacobi iteration $A_1=D$ $A_2=L+U$
 - A_1^{-1} is easy to compute (1/ entries along diagonal)
 - This is easy to compute with the FMM
 - At element level $(x_{k+1})_i = a_{ii}^{-1} \left(b_i - \sum_{j \neq i} a_{ij}(x_k)_j \right)$
- Other classical iterations (Gauss-Seidel, SOR, etc. are harder to write in a way that FMM can be used).

Krylov methods

- Different class of methods
- Do not involve an iteration matrix
- Motivation: Say in functional space we are at a point x_0 and we want to reach the solution x^*
- Can do it by taking steps along some directions
- Method of steepest descent
- Define function $f(x)$ $f(x) = \frac{1}{2}x^tAx - b^t x$
- So minimum of $f(x)$ is attained at $\nabla f(x)=0$

$$\nabla f = \frac{1}{2}(A + A^t)x - b = 0$$

$$\frac{1}{2}(A + A^t)x = b \text{ or } Ax = b$$

for symmetric A

Steepest descent

- So finding minimizer of the quadratic form gives a solution
- Start at guess x_0 , take a step along $-\nabla f(x_0)$

$$\nabla f(x_0) = Ax_0 - b = -r_0$$

$$x_1 = x_0 + \alpha r_0$$

- How big should the step α be?
- We can find α that minimizes $f(x_0 + \alpha r_0)$

$$\frac{d}{d\alpha} f(x_0 + \alpha r_0) = r_0 \cdot \nabla f|_{x=x_0+\alpha r_0}$$

- So if α should be chosen so that r_0 is orthogonal to ∇f

- Can show
$$\alpha = \frac{r_0^T r_0}{r_0^T A r_0}$$

Steepest Descent method

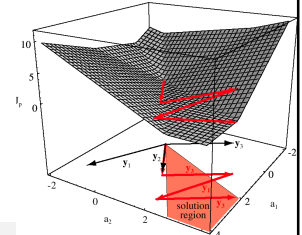
- Putting things together
- Requires two matrix vector products with A per iteration

$$r^{(i)} = b - Ax^{(i)},$$

$$\alpha^{(i)} = \frac{r^{(i)T} r^{(i)}}{r^{(i)T} A r^{(i)}},$$

$$x^{(i+1)} = x^{(i)} + \alpha^{(i)} r^{(i)}$$

- Can reduce one
- Can convert last equation to
$$r_{i+1} = r_i - \alpha_i A r_i$$
- only need calculate $A r_i$
- What about convergence?
- Can show at k th iteration



$$\|e_k\|_A \leq \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right)^k \|e_0\|_A \quad \text{where } \kappa(A) = \lambda_{max} / \lambda_{min}$$

Conjugate Gradient

- Instead of minimizing each time along gradient, choose a set of basis directions to minimize along
- Choose these directions to be from the Krylov subspace
- Definition: Krylov subspace $\mathcal{K}_k = \text{span}(r_0, A r_0, \dots, A^{k-1} r_0)$
- Definition: Energy or A -norm of a vector $\|x\|_A = (x^T A x)^{1/2}$
- Idea of conjugate gradient method
 - Generate Krylov subspace directions, and take a step that minimizes the A norm of the residual along this direction
- Let search direction be at step $k+1$ be d_{k+1}
- We require $f(x_k + \alpha_{k+1} d_{k+1})$ is minimized along direction
- Conjugacy property $d_{k+1}^T A K^k = 0$

CG Minimization

- We want to make the current solution's to be the one whose error is minimized in the energy norm
- Express the solution in terms of Krylov basis
$$x^{(k)} = \alpha_1 v_1 + \dots + \alpha_k v_k = V_k \alpha,$$
- Here V is the $N \times k$ matrix with the Krylov vectors up to current iteration as columns

$$\begin{aligned} E(x^{(k)}) &= (x^{(k)} - x^*)^T A (x^{(k)} - x^*) \\ &= (V_k \alpha - x^*)^T A (V_k \alpha - x^*) \end{aligned}$$

- Set $dE/d\alpha = 0$

CG Minimization

- Computing the derivative

$$2V_k^T AV_k \alpha - 2V_k^T Ax^* = 0$$

If the columns of V_k are orthogonal in the A inner product, then $V_k^T AV_k = D_k$, a diagonal matrix, so

$$\alpha_i = \frac{v_i^T b}{v_i^T A v_i}, i = 1, \dots, k.$$

- Observation: As iteration proceeds, because of the orthogonality, the earlier components do not change.
 - So all we need to do is compute the new entry α_i
 - Practical implementation is a little different

CG Algorithm

- Each iteration requires computation of one matrix vector product Ad_i
- Guaranteed to converge after N iterations (for exact arithmetic)
- So FMM guaranteed to solve equations in $O(N^2)$ time
- Method does not require storage of any of the Krylov basis vectors
- Trivial to add preconditioning
- Implemented in Matlab as **pcg**

$$d_{(0)} = r_{(0)} = b - Ax_{(0)},$$

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \quad ($$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)},$$

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)},$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}},$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}.$$

PCG

- Pcg: Preconditioned conjugate gradients method
- $x = \text{pcg}(A, b)$
 - $\text{pcg}(A, b, \text{tol})$
 - $\text{pcg}(A, b, \text{tol}, \text{maxit})$
 - $\text{pcg}(A, b, \text{tol}, \text{maxit}, M)$
 - $\text{pcg}(A, b, \text{tol}, \text{maxit}, M1, M2, x0)$
- $[x, \text{flag}] = \text{pcg}(A, b, \dots)$
- $[x, \text{flag}, \text{relres}] = \text{pcg}(A, b, \dots)$
- $[x, \text{flag}, \text{relres}, \text{iter}] = \text{pcg}(A, b, \dots)$
- $[x, \text{flag}, \text{relres}, \text{iter}, \text{resvec}] = \text{pcg}(A, b, \dots)$
- Description $x = \text{pcg}(A, b)$ attempts to solve the system of linear equations $A^*x=b$ for x . The n -by- n coefficient matrix A must be symmetric and positive definite, and should also be large and sparse.
- The column vector b must have length n .
- A can be a function handle afun such that $\text{afun}(x)$ returns A^*x .**

Convergence of CG

- If there are k distinct eigenvalues of A , then CG converges in at most k iterations to the exact solution (in exact arithmetic)
- Usually we go to $\|b - Ax_k\|_2 \leq \eta \|b\|_2$

$$\|x_* - x_m\|_A \leq 2 \left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right]^m \|x_* - x_0\|_A.$$

- Converges quickly
- Compare with steepest descent

$$\|e_k\|_A \leq \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right)^k \|e_0\|_A \quad \text{where } \kappa(A) = \lambda_{\max} / \lambda_{\min}$$

Non symmetric matrices

- Can't apply CG to nonsymmetric matrices
- One simple solution – CGNR j
 - convert system to a symmetric system
$$A'Ax=A'b$$
- However we will need two matrix vector multiplies per iteration
- Also, if A is poorly conditioned then $A'A$ is even more poorly conditioned (condition number is squared)
- However the method requires no storage of Krylov basis.

GMRES

- Instead of requiring minimization along conjugate direction, minimize residual in a subspace
- Krylov subspace $\bar{K}_k = \text{span}(r_0, Ar_0, \dots, A^{k-1}r_0)$
- Require x to minimize $\|b - Ax\|_2 \quad \forall x \text{ in } x_0 + K^k$
- Construct basis


```

w(0) = Av(0)
for k = 1, ..., i
    w(k) = w(k) - (w(k), v(k))v(k)
end
v(k+1) = w(k) / ||w(k)||
            
```
- Then require that each $w^{(k)} = w^{(k)} + \alpha_k v^{(k)} + \dots + \alpha_k v^{(k)}$, satisfies minimum $\|b - Aw^{(k)}\|$
- Can be done by simple minimization
- Implemented as a black-box in Matlab

Bi Conjugate methods

- Since A is non symmetric, A^t and A have different eigenvectors
- Form the Krylov basis of A^t for some vector g (usually equal to r_0) $\bar{K}_k = \text{span}(r_0, A^t r_0, \dots, (A^t)^{k-1} r_0)$
- Minimize successively along directions so that $p_k A d_i = 0$
- Storage cost is lower ... Bi-CGSTAB

Eigenvalue problems: Power Iteration

- Computes the dominant eigen value and eigen vector of a given matrix A .
- $x_k = Ax_{k-1}$ converges to the dominant eigen vector v_1
- Ratio of values of given component of x_k from one iteration to next converges to the dominant eigen value λ_1 .
- If two eigen values are equal may converge to a linear combination of the corresponding eigen vectors.
- To prevent overflow or underflow use Normalized power iteration. $x_k = Ax_{k-1} \quad x_k = x_k / \|x_k\|_\infty$.
- Can do power iteration with shift to increase convergence rate.

Krylov methods for Eigenvalues

- Arnoldi methods → Implemented in arpack
- Available in Matlab as eigs

`d = eigs(A)` returns a vector of A's six largest magnitude eigenvalues. `[V,D] = eigs(A)` returns a diagonal matrix D of A's six largest magnitude eigenvalues and a matrix V whose columns are the corresponding eigenvectors. `[V,D,flag] = eigs(A)` also returns a convergence flag. If flag is 0 then all the eigenvalues converged; otherwise not all converged. `eigs(A,B)` solves the generalized eigenvalue problem $A^*V == B^*V^*D$. B must be symmetric (or Hermitian) positive definite and the same size as A. `eigs(A,[],...)` indicates the standard eigenvalue problem $A^*V == V^*D$. `eigs(A,k)` and `eigs(A,B,k)` return the k largest magnitude eigenvalues. `eigs(A,k,sigma)` and `eigs(A,B,k,sigma)` return k eigenvalues based on sigma, which can take any of the following values: scalar (real or complex, including 0) The eigenvalues closest to sigma.

Research: Preconditioning for the FMM

- In all these methods (especially at the large sizes where the FMM is used), iterative methods are slowly convergent
- Unless iteration converges relatively quickly (less than 100 iterations), it may not be practically useful
- Condition number is important in quantifying convergence
 - Project for estimating condition number
- Most preconditioners were invented to speed up sparse matrix computations
- Take $O(N^2)$ time for dense matrices (or too slow)
- Goal: Fashion good preconditioners for the FMM

Preconditioner cost

- To be effective with the FMM the preconditioner cost should be $O(N \log N)$ or less
- Otherwise we negate the advantage of the FMM
- Our recent research two preconditioners for FMM
 - First, for highly oscillatory kernel --- multiple scattering with the Helmholtz equation
 - Second, we accelerate a highly successful preconditioner developed by Faul, Powell and Goodsell (2005), but which has a complexity of $O(N^2)$ to $O(N \log N)$.