Scientific Computing (PHYS 2109/Ast 3100 H)
II. Numerical Tools for Physical Scientists

SciNet HPC Consortium
University of Toronto

Winter 2014

Lecture 13: Numerical Linear Algebra

Part I - Theory

- Solving $Ax = b$
- System Properties
- Direct Solvers
- Iterative Solvers
- Dense vs. Sparse matrices

Lecture 13: Numerical Linear Algebra

Part I - Theory

- Solving $Ax = b$
- System Properties
- Direct Solvers
- Iterative Solvers
- Dense vs. Sparse matrices

Part II - Application

- Using packages for Linear Algebra
- BLAS & LAPACK
- etc..

Numerical Linear Algebra

Many algorithms require the solution of a sequence of structured linear systems, such as implicit time-marching schemes, Newton's method, gradient based optimization, statistics, data fitting.. etc.

- A significant amount of memory usage and computation time is spent constructing and solving these systems.
- Many methods and approaches exist:
  - direct vs. iterative
  - sparse vs. dense
- Choice of method depends on nature of system being solved and can drastically affect solution time and accuracy.
- **DON'T** program your own, use a library

Solving Linear Systems: $Ax = b$ solve for x

Sets of linear equations: don't invert

- $Ax = b$ implies $x = A^{-1}b$
- Mathematically true, but numerically, inversion:
  - is slower than other solution methods
  - is numerically much less stable
  - ruins sparcity (**huge** memory disadvantage for, eg, PDEs on meshes)
  - loses any special structure of matrix A

Easy systems to solve

- We'll talk about methods to solve linear systems of equations
- Will assume nonsingular matricies (so there exists a unique solution)
- But some systems much easier to solve than others. Be aware of "nice" properties of your matricies!

Diagonal Matrices

- (generally called D, or Λ)
- Ridiculously easy
- Matrix multiplication - just d_i x_i

$$\begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \cdots & \\ & & & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$x_i = \frac{b_i}{d_i}$$

Upper Triangular Matrices

- Generally called U
- "Back Substition": solve (easy) last one first
- Use that to solve previous one, etc.
- Lower triangular (L): "Forward substitution", same deal.

$$\begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ & u_{2,2} & \cdots & u_{2,n} \\ & & \cdots & \vdots \\ & & & u_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$x_n = \frac{b_n}{u_{n,n}}$$

$$x_{n-1} = \frac{b_n - u_{n-1,n} x_n}{u_{n-1,n-1}}$$

$$\vdots$$

Orthogonal matrices

- Generally called Q
- Columns (rows) are orthogonal unit vectors
- Transpose is inverse!
- *That* inverse I'll let you compute.
- Orthogonal matrices are numerically very nice - all row, col vectors are same "length".

$$
\begin{aligned}
Q^T Q &= I \\
Q\mathbf{x} &= \mathbf{b} \\
Q^T Q\mathbf{x} &= Q^T\mathbf{b} \\
\mathbf{x} &= Q^T\mathbf{b}
\end{aligned}
$$

Symmetric Matrices

- No special nomenclature
- Half the work; only have to deal with half the matrix
- (I'm assuming real matrices, here; complex: Hermetian)

$$A^T = A$$

$$a_{i,j} = a_{j,i}$$

Symmetric Positive Definite

- Very special but common (covariance matricies, some PDEs)
- Always non-singular
- All eigenvalues positive
- Numerically very nice to work with

$$A^T = A$$

$$\mathbf{x}^T A \mathbf{x} > 0$$

$$A = LL^T$$

Structure matters

- Find structure in your problems
- If writing equations in slightly different way gives you nice structure, do it
- Preserve structure when possible

System Properties

Conditioning

- A problem is said to be inherently ill-conditioned if any small perturbation in the initial conditions generates huge changes in the results
- Say, calculating $f(x)$: if

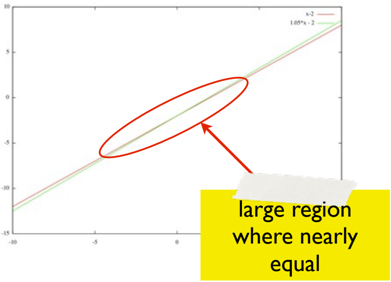
$$\frac{||f(x+\delta x)||}{||f(x)||} \gg \frac{||\delta x||}{||x||}$$

then the problem is inherently hard to do numerically (or with any sort of approximate method)

Conditioning

- In matrix problems, this can happen in nearly singular matricies - nearly linearly dependant columns.
- Carve out strongly overlapping subspaces
- Very small changes in b (say) can result in hugely different change in x

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.05 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$



Condition number

- Condition number can be estimated using "sizes" (matrix norms) of A, inverse of A.
- Lapack routines exist: ___CON
- Relative error in x can't be less than condition number * machine epsilon.

$$\kappa(A) = ||A|| \cdot ||A^{-1}||$$

$$\frac{||\delta x||}{||x||} < \kappa(A) \frac{||\delta b||}{||b||}$$

Residuals

- Computational scientists have over 20 words for "numerical error"
- Absolute, relative error - error in x.
- **Residual**: answer in result provided by erroneous x - error in b.
- Which is more important is entirely problem dependant

Residuals

- Good linear algebra algorithms (and implementations) should give residuals no more than (some function of size of matrix) x (machine epsilon)

- And errors in x no more than condition number times that.

- An exact solution to a nearby problem

- Bad algorithms/implementations will depend on sqrt(machine epsilon) or worse, and/or will be matrix dependant (eg, LU without pivoting).

Pivoting

- The diagonal elements we use to “zero out” lower elements are called pivots.
- May need to change pivots, if for instance zeros appear in wrong place
- Matrix might be singular, or fixed by reordering
- PLU factorization

$$A = \begin{pmatrix} 0 & a & b \\ 0 & 0 & c \\ d & e & f \end{pmatrix}$$

Eigenproblems

- Tells a great deal about the structure of a matrix
- How it will act on a vector: project onto its eigenvectors, mutiply by eigenvalues.
- Goal is a complete decomposition:

$$A\mathbf{x} = \lambda\mathbf{x}$$

$$A\left[\begin{array}{c|c|c|c} x_1 & x_2 & \ldots & x_n \end{array}\right] = \left[\begin{array}{c|c|c|c} x_1 & x_2 & \ldots & x_n \end{array}\right]\begin{bmatrix} \lambda & & & \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \end{bmatrix}$$

SciNet compute • calcul CANADA

Eigenvalue Decomposition

- For square matrix
- "Similarity Transform"
- No restrictions on the structure of X
- Can only happen if there are a full set of eigenvectors.
- Diagonalizability: N non-null eigenvectors;
- Invertability: N non-zero eigenvalues

$$A \left[\begin{array}{c|c|c|c} x_1 & x_2 & \dots & x_n \end{array} \right] = \left[\begin{array}{c|c|c|c} x_1 & x_2 & \dots & x_n \end{array} \right] \begin{bmatrix} \lambda & & & \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \end{bmatrix}$$

$$AX = X\Lambda$$

$$A = X\Lambda X^{-1}$$

Solve Ax=b

Gaussian Elimination

- For general square matrices (can't exploit above properties)
- We all learned this in high school:
  - Subtract off multiples of previous rows to zero out below-diagonals
  - Back-subsitute when done

$$\begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -2 & 2 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ & 2.5 & 5 \\ & 3.4 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ -0.5 \\ 2.6 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ & 2.5 & 5 \\ & & -0.8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ -0.5 \\ 3.28 \end{pmatrix}$$

Gaussian Elimiation = LU Decomposition

- With each stage of the elimination, we were subtracting off some multiple of a previous row

- That means the factored U can have the same multiple of the row added to it to get back to A

- Decomposing to give us A = L U

$$\begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -2 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -2 & 2 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -2 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1 & & \\ +\frac{1}{2} & 1 & \\ -\frac{1}{5} & & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ & 2.5 & 5 \\ & 0.6 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ 5 & -1 & 5 \\ -2 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ +\frac{1}{5} & +\frac{6}{25} & 1 \end{pmatrix} \begin{pmatrix} 10 & -7 & 0 \\ & 2.5 & 5 \\ & & 4.8 \end{pmatrix}$$

$$A = LU$$

Solving is fast with LU

- Once have A = LU ($O(n^3)$ steps) can solve for x quickly ($O(n^2)$ steps)
- Can solve for same A with different b very cheaply
- Backsubstitute, then forward substitute

$$
\begin{aligned}
A\mathbf{x} &= \mathbf{b} \\
LU\mathbf{x} &= \mathbf{b} \\
L(\mathbf{y}) &= \mathbf{b} \\
y &= \mathrm{Backsubst}(L, \mathbf{b}) \\
U\mathbf{x} &= \mathbf{y} \\
x &= \mathrm{Forwardsubst}(U, \mathbf{y})
\end{aligned}
$$

$\mathbf{Ax} \sim \mathbf{b}$

A x ~ b : QR factorizations

- Not all Ax=b s can be solved; consider an overdetermined system (data fitting).
- LU won't even work on non-square systems.
- What to do?

$$\begin{pmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ x_1^3 & x_1^2 & x_1 & 1 \\ \dots & & & \\ x_n^3 & x_n^2 & x_n & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_n \end{pmatrix}$$

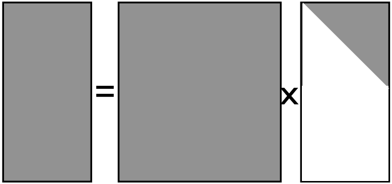
Minimize residual: Residual not in Range(A)

- Want to project out residual somehow
- Normal equations
- Much of linear algebra is decompositions into useful forms

$$
\begin{aligned}
\mathbf{r}^2 &= ||\mathbf{b} - A\mathbf{x}||_2^2 \\
&= (\mathbf{b} - A\mathbf{x})^T(\mathbf{b} - A\mathbf{x}) \\
&= \mathbf{b}\cdot\mathbf{b} - 2\mathbf{b}^T A\mathbf{x} + \mathbf{x}^T A^T A\mathbf{x} \\
0 &= -2\mathbf{b}^T A + 2\mathbf{x}^T A^T A \\
(A^T A)\mathbf{x} &= A^T\mathbf{b}
\end{aligned}
$$

QR decomposition

- All matricies can be decomposed into QR, even m×n, m>n
- Bottom half of R is necessarily empty (below diagonal)
- All columns in Q are orthogonal bases of m-d space, and R is the combination of them that makes up A



Normal equations with QR are easy

- Now this is fairly straightforward
- End up with (Rx) -- forward solve -- equal to matrix-vector product.
- Done!

$$
\begin{aligned}
(A^T A)\mathbf{x} &= A^T\mathbf{b} \\
R^T Q^T Q R\mathbf{x} &= R^T Q^T \mathbf{b} \\
R^T R\mathbf{x} &= R^T Q^T \mathbf{b} \\
R\mathbf{x} &= Q^T \mathbf{b}
\end{aligned}
$$

SciNet compute • calcul CANADA

Iterative Methods

Iterative Methods

- So far, have dealt solely with direct methods.
- Solution takes one (long) step, then answer is complete, as exact as matrix/method allows.
- Other approach; take successive approximations, get closer.
- Typically converge to machine accuracy in much less time than direct, esp for large matricies

Krylov Subspaces

- Krylov subspace: repeated action on b by A.
- For sufficiently large n, final term should converge to eigenvector with largest eigenvalue
- But slow, and only one eigenvalue?

$$A\mathbf{x} = \mathbf{b}$$

$$\mathcal{K} = \left[\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \cdots, A^{n-1}\mathbf{b}\right]$$

Krylov Subspaces

- Can orthogonalize (Gram Schmidt, Householder) to project out other components
- Should give approximations to eigenvectors (random b)
- But not numerically stable

$$A\mathbf{x} = \mathbf{b}$$

$$\mathcal{K} = \left[\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \cdots, A^{n-1}\mathbf{b}\right]$$

Arnoldi Iteration

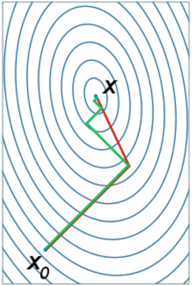
- Stabilized orthogonalization
- Becomes Lanczos iteration for symmetric A
- Orthogonal projection of A onto the Krylov subspace, H
- H is of modest size, can have eigenvalues calculated
- Note: Only requires matrix-vector, vector-vector products
- GMRES: Arnoldi iteration for solving Ax=b

$$
\begin{aligned}
& q_1 \leftarrow e_1 \\
& \text{for } j \in [1, k-1]: \\
& \qquad h_{j,k-1} \leftarrow q_j^T q_k \\
& \qquad q_k \leftarrow q_k - h_{j,k-1} q \\
& h_{k,k-1} \leftarrow ||q_k|| \\
& q_k \leftarrow \frac{q_k}{h_{k,k-1}}
\end{aligned}
$$

Iterative Ax=b solvers: Conjuate Gradient

- SPD matrices, works particularly well on sparse systems

- "Steepest Descent", but only on conjugate (w/rt A) directions: no "doubling back"



http://en.wikipedia.org/wiki/Conjugate_gradient_method

Iterative Solvers - Summary

- GMRES (generalized minimal residual method)
- BI-CGSTAB (bi conjugate gradient stabalized)
- Almost always need preconditioning for good preformance
  - Jacobi
  - ILU
  - SOR
  - AMG
  - Schur, Schwarz (parallel)

Iterative Solvers - Summary

- GMRES (generalized minimal residual method)
- BI-CGSTAB (bi conjugate gradient stabalized)
- Almost always need preconditioning for good preformance
  - Jacobi
  - ILU
  - SOR
  - AMG
  - Schur, Schwarz (parallel)

More Information

"Iterative methods for sparse linear systems" - Yousef Saad
http://www-users.cs.umn.edu/ saad/

Sparse Matrices

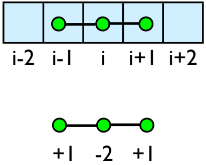
Sparse Matricies

- So far, we've been assuming our matrices are dense; there are numbers stored for every entry in matrix.
- This is indeed often the case, but it's also often that huge numbers of the entries are zero: some roughly constant number of entries per row, much less than n.
- Difference between n^2 and n can be huge if $n \sim 10^6$; difference between doing and not doing the problem.
- Happens particularly often in discretizing PDEs.

Discretizing Derivatives

$$\left.\frac{d^2q}{dx^2}\right|_i \approx \frac{q_{i+1} - 2q_i + q_{i-1}}{\Delta x^2}$$

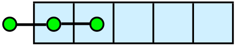
- Done by finite differencing the discretized values
- Implicitly or explicitly involves interpolating data and taking derivative of the interpolant



$$\frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & 1 & -2 & 1 & & & \\ & & & 1 & -2 & 1 & & \\ & & & & & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & \end{pmatrix}\mathbf{q}_i$$

Boundary Conditions

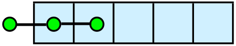
$$\left.\frac{d^2q}{dx^2}\right|_i \approx \frac{q_{i+1} - 2q_i + q_{i-1}}{\Delta x^2}$$



- What happens when stencil goes off of the end of the box?
- Depends on how you want to handle boundary conditions.
- Typically easiest to have extra points on end, set values to enforce desired BCs.

Boundary Conditions

$$\left.\frac{d^2q}{dx^2}\right|_i \approx \frac{q_{i+1} - 2q_i + q_{i-1}}{\Delta x^2}$$



- Dirichlet (fixed value) boundary conditions: just have 1 on diagonal, 0 elsewhere, keeps value there constant.
- Neumann (derivitave) bcs: requires more manipulation of the equations.

Inverses destroy sparsity

- For sparse matrices like above, LU decompositions may maintain much sparsity (particularly if banded, etc)

$$\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & -1 & 1 & \\ & & & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & 1 & -1 & \\ & & & 1 & -1 \\ & & & & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 5 & 4 & 3 & 2 & 1. \\ 4 & 4 & 3 & 2 & 1. \\ 3 & 3 & 3 & 2 & 1. \\ 2 & 2 & 2 & 2 & 1. \\ 1 & 1 & 1 & 1 & 1. \end{pmatrix}$$

- Inverses in general are full

- For large n, difference beween cn and n^2 huge.

Sparse (banded) LU

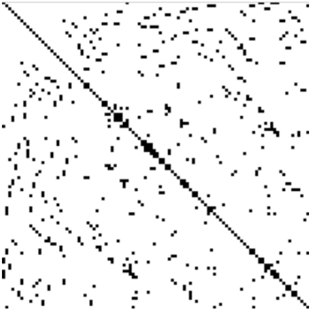
- If entries only exist within a narrow band around diagonal, then row, column operations fast.
- May get significant "fill in" depending on exact structure of matrix
- (This is artificially good example)

$$\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} =$$

$$\begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & -1 & 1 & \\ & & & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & 1 & -1 & \\ & & & 1 & -1 \\ & & & & 1 \end{pmatrix}$$

Sparsity patterns

- Sparse matrices can have arbitray sparsity patterns
- Typically need at less than 10% nonzeros to make dealing with sparse matricies worth it.
- Half zeros - typically just store full matrix.



http://en.wikipedia.org/wiki/File:Finite_element_sparse_matrix.png

Common Sparse Matrix Formats:

- CSR (Compressed Sparse Row): Just join all the nonzeros in rows together, with pointers to where each starts, and (similar sized) array of column for each value
- CSC (Compressed Sparse Column): Same, but flip row/column
- Banded: just store diagonals +/- some bandwidth
- Many many more.

Conclusions - part I

- Linear algebra pops up everywhere, even if you don't notice
- Stats, data fitting, graph problems, PDE/ODE solves, sig. processing
- Exploit structure in your matrices
- Choose method based on system properties
- Don't ever directly invert a matrix
- Pick the solution method that exploits structure in your matrices

Conclusions - part I

- Linear algebra pops up everywhere, even if you don't notice
- Stats, data fitting, graph problems, PDE/ODE solves, sig. processing
- Exploit structure in your matrices
- Choose method based on system properties
- Don't ever directly invert a matrix
- Pick the solution method that exploits structure in your matrices

Next Lecture

- Don't re-invent the wheel.
- There are many very highly tuned packages for any sort of problem that can be cast into matrices and vectors.
- BLAS, LAPACK, etc...