CS590C: Introduction to Computational Science

Lecture 9: Linear Algebra.

This lecture is intended to familiarize you with:

- Basic problems in linear algebra.
- Sparse linear system solvers.

Problems in Computational Science and Engineering

Computational science problems can often be reduced to solving one or a sequence of the following standard linear algebra problems:

- ❑ Solving linear systems of equations: $Ax = b$; Here A is a given n-by-n nonsingular real or complex matrix, b is a given column vector with n entries, and x is a column vector with n entries we wish to compute.

- ❑ Least squares problems: compute the x which minimizes $||Ax - b||$ where A is m-by-n, b is m-by-1, and x is n-by-1. $||r||$ is called the 2-norm of the vector r. If $m>n$, so we have more equations than unknowns, the system is called overdetermined.

- ❑ Eigenvalue problems: Given an n-by-n matrix A, find an n-by-1 nonzero vector x and a scalar λ so that $Ax = \lambda x$.

For example, linear equations often arise when solving ordinary or partial differential equations numerically; the linear system must be solved to advance the solution by a time step. Least squares problems arise in fitting curves or surfaces to experimental data. Eigenvalue problems arise when analyzing vibrations.

Problems in Linear Algebra

The choice of algorithm depends on three things:

- The **structure of A**.
Whether it is dense or sparse or diaginal or Toeplitz.
For example, a dense LU decomposition takes $O(n^3)$ steps, a sparse LU decomposition takes $O(n^{3/2})$ operations for 2-D problems and $O(n^{4/3})$ for 3-D problems. Sparse matrices treated as banded dense matrices take somewhere between $O(n^3)$ and this complexity.

- The **desired accuracy** limits:
For example, if one needs to solve to 16 decimals of accuracy, one might use a careful and expensive implementation of Gaussian elimination, but if one only needs 3 decimal digits, a few iterations of a much cheaper iterative method like conjugate gradients might suffice.

- The **computer system**:
Precision of the machine, memory system, available memory, ratio of available memory to processor speed all have profound impact on selected algorithm. (Eg. Mat-mat blocked formulations).

Software for Dense Linear Algebra.

BLAS: Basic Linear Algebra Subroutines

Three levels of routines:

- ❏ BLAS1: Vector-vector operations
- ❏ BLAS2: Matrix-vector operations
- ❏ BLAS3: Matrix-Matrix Operations

EISPACK: Eigenvalues and Eigenvectors

LINPACK: collection of Fortran subroutines that analyze and solve linear equations and linear least-squares probles. The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems.

LAPACK: above + more efficient operations on shared-memory parallel computers

ScaLAPACK: LAPACK on distributed memory parallel computers.

BLAS Subroutines

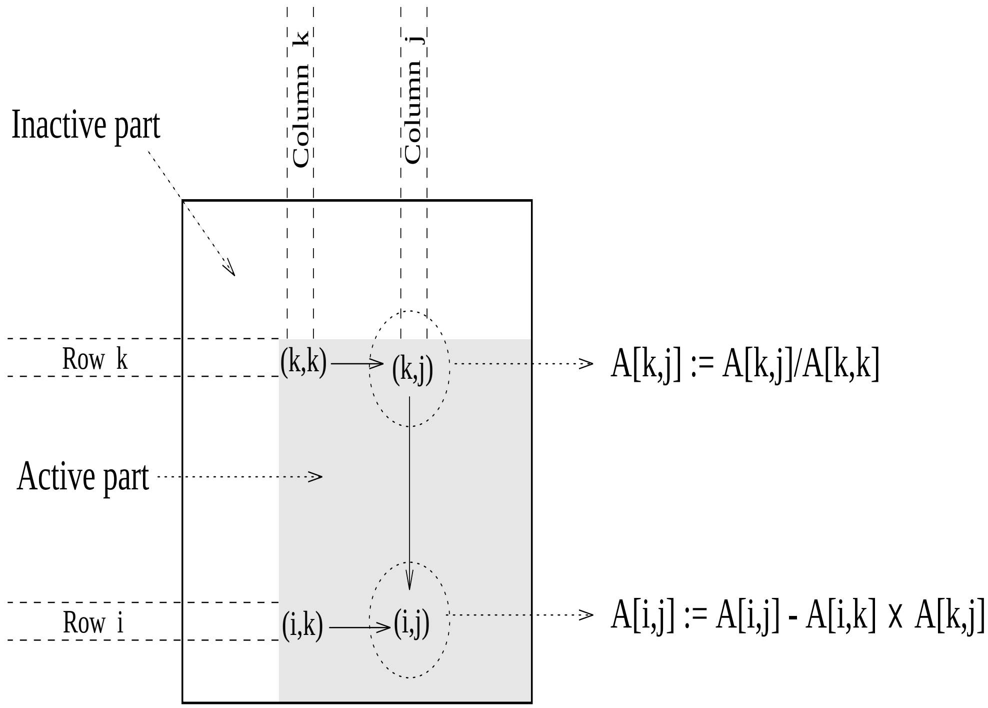
Operation	Definition	FLOPS	Mem. Refs	*q*
saxpy	$y_i = ax_i + y_i$ $(i=1\text{-}n)$	$2n$	$3n + 1$	$2/3$
mat-vec	$y = Ax + y$	$2n2$	$n2+3n$	2
mat-mat	$C = A.B + C$	$2n3$	$4n2$	$n/2$

Here, q is the ratio of FLOPS performed to number of memory accesses. What does q tell us?

Gaussian Elimination on Dense Matrices

```
1.      procedure GAUSSIANELIMINATION (A, b, y)
2.      begin
3.      for k := 0 to n-1 do
        /* Outer loop */
4.      begin
5.              for j := k+1 to n-1 do
6.                      A[k,j] := A[k,j]/A[k,k];  /* Division step */
7.              y[k] := b[k]/A[k,k];
8.              A[k,k] := 1;
9.              for i := k+1  to n-1  do
10.             begin
11.                     for j := k+1  to n-1  do
12.                             A[i,j] := A[i,j]-A[i,k]x A[k,j];  /* Elimination step */
13.                     b[i] := b[i] - A[i,k]x y[k];
14.                     A[i,k] := 0;
15.             endfor; /* Line 9 */
16.     endfor;  /* Line 3 */
17.end GAUSSIANELIMINATION
```

Gaussian Elimination on Dense Matrices



Iterative Solvers

- ❏ Another technique for solving $Ax = b$.
- ❏ These are memory optimal (use $O(n^2)$ memory).
- ❏ If convergence properties are good, these are beneficial since each iteration only takes $O(n^2)$ time.
- ❏ Start with arbitrary solution; generate a sequence of solutions that will (hopefully) converge to the solution.
- ❏ Typically used with large, sparse linear systems.
- ❏ Easy to parallelize.
- ❏ Often need preconditioners to accelerate their convergence.

Iterative Solvers

- Given A, find M and N so that
  $A = M - N$
  M is invertable
  M^{-1} can be computed easily

- Then:
  $(M-N)x = b$
  $Mx = Nx + b$
  $x = M^{-1}Nx + M^{-1}b$

- Let $B = M^{-1}N$ and $g = M^{-1}b$
  $x = Bx + g$

- So try the following:
  x^0 = arbitrary n-dimensional vector
  $x^{m+1} = Bx^m + g$, $m = 0, 1, ...$

- If $y = \lim x^m$; m -> infinity then $y = A^{-1}b$.

- Can prove that this happens if spectral radius of B < 1.

Commonly used Ms and Ns

Consider $A = D - L - U$

Jacobi Method

$M = D$
$Ax = b$
$(D - L - U)x = b$
$Dx = (L+U)x + b$
$x = D^{-1}(L+U)x + D^{-1}b$

Gauss Siedel Method

$M = D - L$
$Ax = b$
$(D - L - U)x = b$
$(D - L)x = Ux + b$
$x = (D - L)^{-1}Ux + (D - L)^{-1}b$

Successive Over-Relaxation (SOR)

$M = 1/w * (D - wL)$
$N = 1/w* ((1-w)D + wU)$
w is called the over-relaxation parameter.
One can prove that for convergence, $0 < w < 2$.

Conjugate Graduent Methods:

❏ Works by computing a Q-orthogonal basis for solution space and finding the coefficients of the solution vectors expansion in this basis.

❏ With exact arithmatic, the algorithm reaches the solution in n steps.

❏ Each y^i is the best choice among all vectors on the subspace.

❏ The change from y^i to y^{i+1} is conjugate to all preceding changes. (Conjugate means perpendicular here)

The Conjugate Gradient Method.

procedure CG (A, b, x, epsilon)

begin

    $k := 0; x_0 := 0; r_0 = b; \rho_0 := ||r_0||^2;$

    while $(\mathrm{sqrt}(\rho_i) > \varepsilon\ ||r_0||)$ do

    begin

        if $(k = 0)$ then $p_1 := r_0$

        else $p_{k+1} := r_k + \rho_k p_k / \rho_{k-1};$

        $k := k+1;$

        $w_k := Ap_k;$

        $\alpha_k := \rho_{k-1} / p_k^T w_k;$

        $x_k := x_{k-1} + \alpha_k\, p_k;$

        $a_k := r_{k-1} - \alpha_k w_k;$

        $\rho_k := ||r_k||^2;$

    endwhile;

    $x := x_k;$

end CG;