Linear Systems & Matrices

Prateek Sharma (prateek@physics.iisc.ernet.in)
Office: D2-08

Extremely Imp.

- used almost everywhere
- multi-D root finding, interpolation, PDEs
- even for nonlinear systems
- exact solution $O(N^3)$ expensive! special forms (tridiagonal) much faster
- iterative methods - very useful in physics
- LAPACK - the linear algebra library

Introduction

N unknowns x_j, $j = 1, 2, \ldots, N$ are related by M equations

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1N}x_N = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2N}x_N = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3N}x_N = b_3$$

$$\cdots \qquad \cdots$$

$$a_{M1}x_1 + a_{M2}x_2 + a_{M3}x_3 + \cdots + a_{MN}x_N = b_M$$

if N=M; good chance of finding unique solution. No unique soln. if

-*row degeneracy*: if one or more rows a linear combination of others

-*column degeneracy*: all eqs. have certain unknowns in same linear comb.

degenerate set of eqs. : *singular*

be careful, numerical solution of close to singular systems is tricky!

if N large: round-off errors can make the intermediate system singular

Matrices

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

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ & \dots & & \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix}$$

M<N, or if eqs. are degenerate, infinitely many solutions; $x=x_p$+any linear combination of (N-M) vectors (null-space)

M>N: *overdetermined*, no solution in general; least sqr. solution s.t., $||b-Ax||_2$ is minimized

we will only deal with NxN matrices

How fast diff. ops.?

- vector-vector dot product $u^T.v$: N Flops
- matrix-vector product: N^2 Flops
- matrix-matrix product: N^3 Flops (can be bettered!)
- matrix inverse, e.g., Gaussian Elim., Gauss-Jordan, LU decomposition: N^3 Flops (can be bettered!)
- solving Ax=b: $O(N^3)$ for exact solution: can be faster for special (e.g., banded) matrices; iterative methods

Gauss-Jordan Elimination

slower than LU decomposition for solving Ax=b but good for finding inverse

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \left[\begin{pmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \end{pmatrix} \sqcup \begin{pmatrix} x_{12} \\ x_{22} \\ x_{32} \\ x_{42} \end{pmatrix} \sqcup \begin{pmatrix} x_{13} \\ x_{23} \\ x_{33} \\ x_{43} \end{pmatrix} \sqcup \begin{pmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ y_{31} & y_{32} & y_{33} & y_{34} \\ y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix} \right]$$

$$= \left[\begin{pmatrix} b_{11} \\ b_{21} \\ b_{31} \\ b_{41} \end{pmatrix} \sqcup \begin{pmatrix} b_{12} \\ b_{22} \\ b_{32} \\ b_{42} \end{pmatrix} \sqcup \begin{pmatrix} b_{13} \\ b_{23} \\ b_{33} \\ b_{43} \end{pmatrix} \sqcup \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \right] \tag{2.1.1}$$

$$[\mathbf{A}] \cdot [\mathbf{x}_1 \sqcup \mathbf{x}_2 \sqcup \mathbf{x}_3 \sqcup \mathbf{Y}] = [\mathbf{b}_1 \sqcup \mathbf{b}_2 \sqcup \mathbf{b}_3 \sqcup \mathbf{1}]$$

$$\mathbf{A} \cdot \mathbf{x}_1 = \mathbf{b}_1 \qquad \mathbf{A} \cdot \mathbf{x}_2 = \mathbf{b}_2 \qquad \mathbf{A} \cdot \mathbf{x}_3 = \mathbf{b}_3$$

$$\mathbf{A} \cdot \mathbf{Y} = \mathbf{1}$$

Ok to interchange rows of A and corresponding rows of b (& I); just reordering eqs.

soln. unchanged if a row in A and b (&I) are replaced by linear comb. of other rows.

Interchanging any two *columns* of **A** gives the same solution set only if we simultaneously interchange corresponding *rows* of the **x**'s and of **Y**.

Gauss-Jordan elimination uses one or more of the above operations to reduce the matrix **A** to the identity matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \xrightarrow{R_1/a_{11}} \begin{bmatrix} 1 & a_{12}/a_{11} & a_{13}/a_{11} & a_{14}/a_{11} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \xrightarrow{\text{relabel}} \begin{bmatrix} 1 & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$R_2 = R_2 - a_{21}R_1$

$R_3 = R_3 - a_{31}R_1$

...

$$\xrightarrow{R_2/a_{22}} \begin{bmatrix} 1 & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & a_{32} & a_{33} & a_{34} \\ & a_{42} & a_{43} & a_{44} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & a_{12} & a_{13} & a_{14} \\ & 1 & a_{23} & a_{24} \\ & a_{32} & a_{33} & a_{34} \\ & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$R_1 = R_1 - a_{12}R_2$

$R_3 = R_3 - a_{32}R_2$

...

$$\begin{bmatrix} 1 & & a_{13} & a_{14} \\ & 1 & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

and so-on

apply same row operations to the RHS; transform A to I,
transformed b/I will be the solution/inverse

Pivoting

run into trouble if we ever encounter a zero element on the (then current) diagonal (pivot)

Pivoting: changing order of rows/columns in the matrix (A) to choose a *desirable* pivot, [& RHS (b), unknown (x)]
Partial pivoting: just rearranging rows of A and b (almost always fine)
Full pivoting: rearranging columns of A and rows of x
w.o. pivoting GJ and other methods are numerically unstable!

simply picking the largest (in magnitude) available element as the pivot is a very good choice
largest after normalizing the biggest coefficient to 1: *implicit* pivoting

$$\begin{bmatrix} 1/100 & 1 & 1/4 \\ 1/10 & 0 & 1/5 \\ 1 & 1 & 1 \end{bmatrix}$$

PP applied only to non-zeroed part of matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 1/10 & 0 & 1/5 \\ 1/100 & 1 & 1/4 \end{bmatrix}$$

Row vs. Column Ops.

row-operations

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

$$(\cdots \mathbf{R}_3 \cdot \mathbf{R}_2 \cdot \mathbf{R}_1 \cdot \mathbf{A}) \cdot \mathbf{x} = \cdots \mathbf{R}_3 \cdot \mathbf{R}_2 \cdot \mathbf{R}_1 \cdot \mathbf{b}$$

$$(\mathbf{1}) \cdot \mathbf{x} = \cdots \mathbf{R}_3 \cdot \mathbf{R}_2 \cdot \mathbf{R}_1 \cdot \mathbf{b}$$

$$\mathbf{x} = \cdots \mathbf{R}_3 \cdot \mathbf{R}_2 \cdot \mathbf{R}_1 \cdot \mathbf{b}$$

interchange of rows (columns) 2 and 4 via left (right) multiplication of following

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

column operations

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

$$\mathbf{A} \cdot \mathbf{C}_1 \cdot \mathbf{C}_1^{-1} \cdot \mathbf{x} = \mathbf{b}$$

$$\mathbf{A} \cdot \mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{C}_2^{-1} \cdot \mathbf{C}_1^{-1} \cdot \mathbf{x} = \mathbf{b}$$

$$(\mathbf{A} \cdot \mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{C}_3 \cdots) \cdots \mathbf{C}_3^{-1} \cdot \mathbf{C}_2^{-1} \cdot \mathbf{C}_1^{-1} \cdot \mathbf{x} = \mathbf{b}$$

$$(\mathbf{1}) \cdots \mathbf{C}_3^{-1} \cdot \mathbf{C}_2^{-1} \cdot \mathbf{C}_1^{-1} \cdot \mathbf{x} = \mathbf{b}$$

$$\mathbf{x} = \mathbf{C}_1 \cdot \mathbf{C}_2 \cdot \mathbf{C}_3 \cdots \mathbf{b}$$

need to store column ops. at each step to get the solution!

row operations are much simpler

Gaussian Elimination

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & 0 & a'_{44} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

reduce the matrix to an upper-triangular form via row ops. & PP

1/3 as expensive as GJ

Backsubstitution:

$$x_4 = b'_4 / a'_{44} \qquad x_3 = \frac{1}{a'_{33}}[b'_3 - x_4 a'_{34}]$$

$$x_i = \frac{1}{a'_{ii}} \left[b'_i - \sum_{j=i+1}^{N} a'_{ij} x_j \right]$$

L-U Decomposition

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b}$$

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

forward substitution

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b}$$

$$y_1 = \frac{b_1}{\alpha_{11}}$$

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$$

$$y_i = \frac{1}{\alpha_{ii}} \left[b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j \right] \qquad i = 2, 3, \ldots, N$$

backward substitution

$$x_N = \frac{y_N}{\beta_{NN}}$$

~N^2 Flops

$$x_i = \frac{1}{\beta_{ii}} \left[y_i - \sum_{j=i+1}^{N} \beta_{ij} x_j \right] \qquad i = N-1, N-2, \ldots, 1$$

LU algorithm

$\alpha_{ii} \equiv 1$; N^2 unknown and N^2 eqs.

$$
\begin{aligned}
i < j: &\quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{ij} = a_{ij} \\
i = j: &\quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ii}\beta_{jj} = a_{ij} \\
i > j: &\quad \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \cdots + \alpha_{ij}\beta_{jj} = a_{ij}
\end{aligned}
$$

Crout's algorithm:

$\alpha_{ii} = 1$, i = 1,...,N

For each j = 1,2,3,...,N

for i = 1, 2, . . ., j

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}.$$

for i = j + 1,j + 2,...,N

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right)$$

pivot

$$
\begin{bmatrix}
\beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\
\alpha_{21} & \beta_{22} & \beta_{23} & \beta_{24} \\
\alpha_{31} & \alpha_{32} & \beta_{33} & \beta_{34} \\
\alpha_{41} & \alpha_{42} & \alpha_{43} & \beta_{44}
\end{bmatrix}
$$

recommended: as same LU decomp.
can be applied to diff RHS

$$\det = \prod_{j=1}^{N} \beta_{jj}$$

Banded Matrices

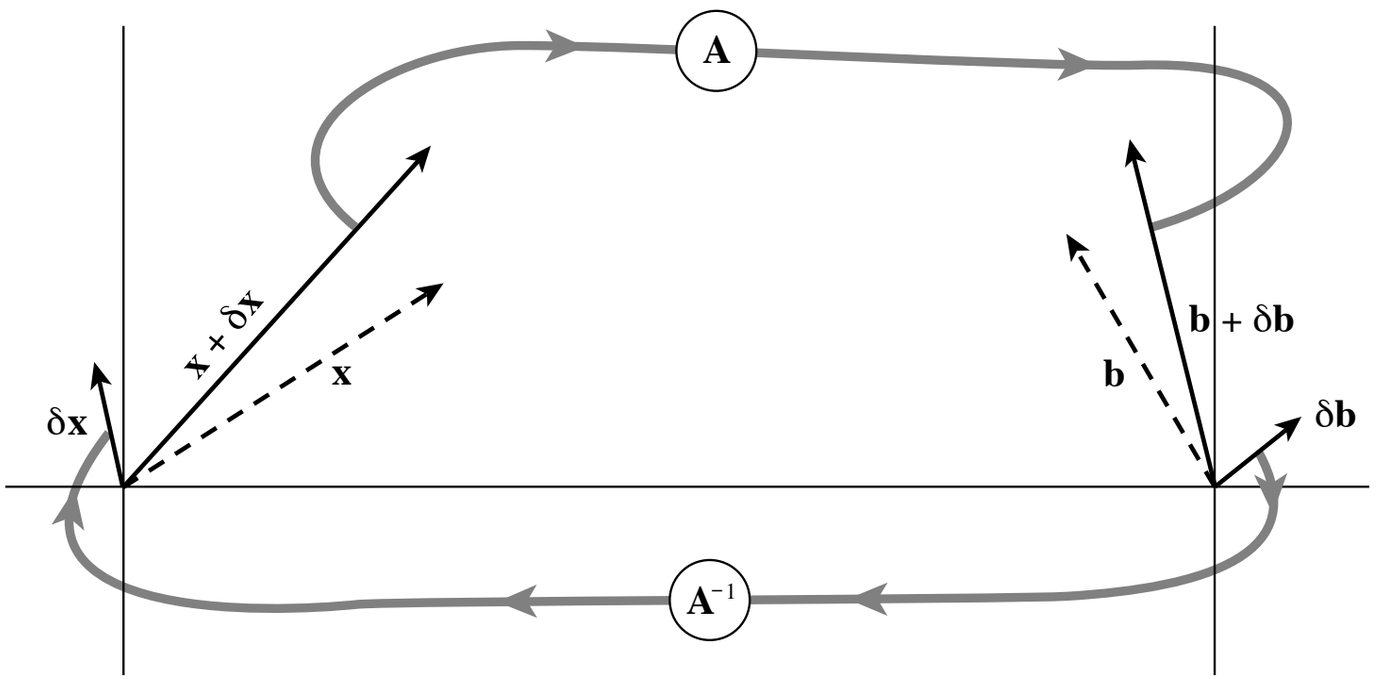
already met tridiagonal systems: O(N); no need of pivoting $\quad |b_j| > |a_j| + |c_j| \qquad j = 1, \ldots, N$

GE & LU decomp. less expensive for banded matrices, e.g., Thomas algorithm for tridiagonal matrices

$$\begin{pmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 5 & 0 & 0 & 0 & 0 \\ 9 & 2 & 6 & 5 & 0 & 0 & 0 \\ 0 & 3 & 5 & 8 & 9 & 0 & 0 \\ 0 & 0 & 7 & 9 & 3 & 2 & 0 \\ 0 & 0 & 0 & 3 & 8 & 4 & 6 \\ 0 & 0 & 0 & 0 & 2 & 4 & 4 \end{pmatrix} \xrightarrow{\text{stored as}} \begin{pmatrix} x & x & 3 & 1 \\ x & 4 & 1 & 5 \\ 9 & 2 & 6 & 5 \\ 3 & 5 & 8 & 9 \\ 7 & 9 & 3 & 2 \\ 3 & 8 & 4 & 6 \\ 2 & 4 & 4 & x \end{pmatrix}$$

do not want to store zeros and waste memory

Iterative Improvement



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

numerical soln. not exact

$$\mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$

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

$$\mathbf{A} \cdot \delta\mathbf{x} = \mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) - \mathbf{b}$$

apply the correction: $O(N^2)$ matrix-vector multiplication & since LU decomp. already available NR recommends this highly

LAPACK

a F90 library of linear algebra routines (http://www.netlib.org/lapack/)

you can download it and install on your computer

```
# -------------------------------------------------------
# Available SIMPLE and DIVIDE AND CONQUER DRIVER routines:
# -------------------------------------------------------

file dgesv.f  dgesv.f plus dependencies
prec double
for  Solves a general system of linear equations AX=B.
gams d2a1

file dsgesv.f  dsgesv.f plus dependencies
prec double / single
for  Solves a general system of linear equations AX=B using iterativement refinement.

file dgbsv.f  dgbsv.f plus dependencies
prec double
for  Solves a general banded system of linear equations AX=B.
gams d2a2

file dgtsv.f  dgtsv.f plus dependencies
prec double
for  Solves a general tridiagonal system of linear equations AX=B.
gams d2a2a
```

a list of libraries:

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

Example

```
      SUBROUTINE DGTSV( N, NRHS, DL, D, DU, B, LDB, INFO )
*  Purpose
*  =======
*  DGTSV  solves the equation
*     A*X = B,
*  where A is an n by n tridiagonal matrix, by Gaussian elimination with
*  partial pivoting.
*  Arguments
*  =========
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
*  NRHS    (input) INTEGER
*          The number of right hand sides, i.e., the number of columns
*          of the matrix B.  NRHS >= 0.
*
*  DL      (input/output) DOUBLE PRECISION array, dimension (N-1)
*          On entry, DL must contain the (n-1) sub-diagonal elements of
*          A.
*
*          On exit, DL is overwritten by the (n-2) elements of the
*          second super-diagonal of the upper triangular matrix U from
*          the LU factorization of A, in DL(1), ..., DL(n-2).
*
*  D       (input/output) DOUBLE PRECISION array, dimension (N)
*          On entry, D must contain the diagonal elements of A.
*
*          On exit, D is overwritten by the n diagonal elements of U.
*
```

be careful & read the routine description;
some of the input matrices are overwritten!

once LAPACK libraries are installed, you can just call the routines

```
#object file
OBJ = modules.o fullimp_cons_reg.o ini_setup.o output.o update.o
#macro definitions
FC = ifort
OPTS = -c -O2
LAPACKHOME = /sw/src/lapack-3.2.1/
#targets
compile:
    ${FC} ${OPTS} modules.f90
    ${FC} ${OPTS} fullimp_cons_reg.f90 ini_setup.f90 output.f90 update.f90
    ${FC} -o run.exe ${OBJ} -L${LAPACKHOME} -llapack -lblas
clean:
    rm -f *.o fort.* *.mod run.exe
```

```
      d(0) = 1.0; d(in+1) = 1.0; du(0) = 1.0; dl(in+1) = 1.0; b(0) = 0.0; b(in+1) = 0.0

      d1=d; dl1=dl; du1=du


      CALL dgtsv( in+2, 1, dl(1:in+1), d(0:in+1), du(0:in), b(0:in+1), in+2, info)

      u=0.0; u(0)=1.0; u(in+1) = 1.0; bu=u
      vt=0.0; vt(1)=-1.0; vt(in)=-1.0

! must remember that the values of dl, etc. are changed after the subroutine!

      CALL dgtsv( in+2, 1, dl1(1:in+1), d1(0:in+1), du1(0:in), bu(0:in+1), in+2, info)

      do i = 0, in+1
        f(i) = b(i) + bu(i)*(b(1)+b(in))/(1.0-bu(1)-bu(in))
      enddo
```