

Lecture 10

- Linear systems
- Iterative techniques
- Singular systems
- Sparse systems
- Sherman-Morrison formula
- Conjugate gradient method

The algebraic, explicit, methods we discussed last class involve increasingly large number of operations when matrices are large. Each of these operations adds round-off error. The errors accumulate.

Since the methods always produce (for non-singular matrices) a solution, one can be easily misled: the end product does not satisfy the system of equations if substituted back into it.



It is therefore desirable to have a method that allows to “improve” a solution. One will never do better than overall machine accuracy, but the point is that in the explicit methods one can do *orders of magnitude worse*.

Iterative methods:

Suppose you have a “slightly wrong solution” to a system of equations:

$$\mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \quad \text{or,} \quad \mathbf{A} \cdot \delta\mathbf{x} = \delta\mathbf{b}$$

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

Improvement  Wrong solution 

If one had found the original solution using LU decomposition, then computing the correction is very economical, since it is tantamount to solving the original system with a different right hand side!

a already LU decomposed

```
subroutine mprove(a,alud,n,np,indx,b,x)
```

```
parameter (nmax=100)
```

```
dimension a(np,np),alud(np,np),indx(n),b(n),x(n),r(nmax)
```

```
real*8 sdp
```

```
do 12 i=1,n
```

```
    sdp=-b(i)
```

```
    do 11 j=1,n
```

```
        sdp=sdp+dbple(a(i,j))*dbple(x(j))
```

```
11    continue
```

```
    r(i)=sdp
```

```
12 continue
```

```
call lubksb(alud,n,np,indx,r)
```

```
do 13 i=1,n
```

```
    x(i)=x(i)-r(i)
```

```
13 continue
```

```
return
```

```
end
```

Compute RHS

Backsubstitute

Update

mprove from Numerical
Recipes

Singular systems:

If the system is singular, the best way to proceed is to find the eigenvalues and eigenvectors and to decompose,

$$\begin{pmatrix} \mathbf{A} \end{pmatrix} = \begin{pmatrix} \mathbf{U} \end{pmatrix} \cdot \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & & \cdots \\ & & & & w_N \end{pmatrix} \cdot \begin{pmatrix} \mathbf{V}^T \end{pmatrix}$$

Where \mathbf{U} and \mathbf{V} are orthogonal matrices
(this decomposition is true even if \mathbf{A} is not square, the \mathbf{U} is column orthonormal).

If \mathbf{A} is square, then the inverse is trivial to compute

$$\mathbf{A}^{-1} = \mathbf{V} \cdot [\text{diag} (1/w_j)] \cdot \mathbf{U}^T$$

Having done a “singular value decomposition” puts us in control of a lot of information about the system. For instance, singularity will manifest itself in some eigenvectors being either zero or very small. One can then decide what to do: it might be that one can, based on physical reasons, eliminate the equations involving the small eigenvalues.

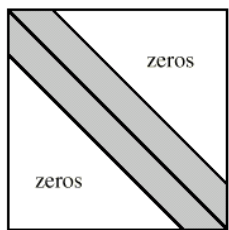
One also finds out if the linear system of equations in question has a solution. The vector of the right hand side should be in the “image” of the solution vector mapped through the matrix A . The presence of the zero eigenvalues shows that such image will not be the entire space of vectors

Singular systems are also of interest with zero right hand side. There one is looking for degenerate subspaces, vectors x that are mapped to zero. Having at hand the eigenvalues one can also construct such vectors easily: they are combinations of the associated eigenvectors.

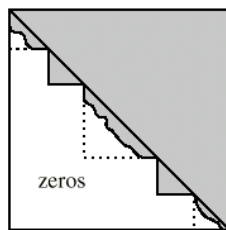
Sparse systems:

Many physical applications (for instance elliptic PDE's) require handling matrices that are very large, but that have many zero components.

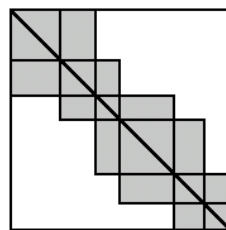
The essence of the techniques for handling these systems is to efficiently encode the information in the matrices in such a way that one does not keep on repeating operations that yield a vanishing result. How this is done depends on the way in which the matrix is sparse



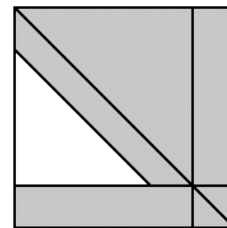
(a)



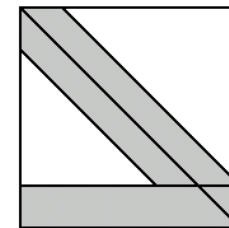
(b)



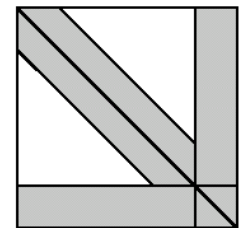
(c)



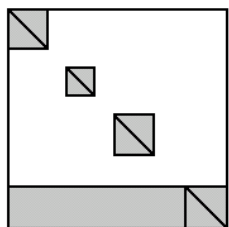
(g)



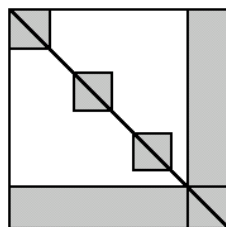
(h)



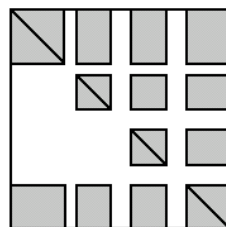
(i)



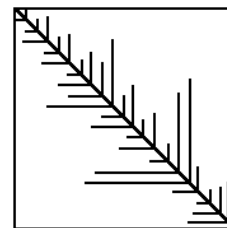
(d)



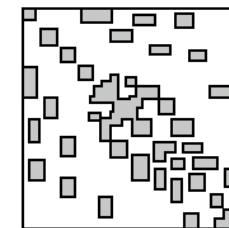
(e)



(f)



(j)



(k)

Although many of these shapes are common enough that there are specific techniques to deal with them, a generic technique is to break the matrix into blocks that one can handle easily,

$$\mathbf{A} = \begin{bmatrix} \mathbf{P} & \mathbf{Q} \\ \mathbf{R} & \mathbf{S} \end{bmatrix} \quad \begin{array}{l} \text{One then} \\ \text{partitions} \\ \text{the inverse} \\ \text{as well} \end{array} \quad \mathbf{A}^{-1} = \begin{bmatrix} \tilde{\mathbf{P}} & \tilde{\mathbf{Q}} \\ \tilde{\mathbf{R}} & \tilde{\mathbf{S}} \end{bmatrix}$$

Then one has,

$$\tilde{\mathbf{P}} = (\mathbf{P} - \mathbf{Q} \cdot \mathbf{S}^{-1} \cdot \mathbf{R})^{-1}$$

$$\tilde{\mathbf{Q}} = -(\mathbf{P} - \mathbf{Q} \cdot \mathbf{S}^{-1} \cdot \mathbf{R})^{-1} \cdot (\mathbf{Q} \cdot \mathbf{S}^{-1})$$

$$\tilde{\mathbf{R}} = -(\mathbf{S}^{-1} \cdot \mathbf{R}) \cdot (\mathbf{P} - \mathbf{Q} \cdot \mathbf{S}^{-1} \cdot \mathbf{R})^{-1}$$

$$\tilde{\mathbf{S}} = \mathbf{S}^{-1} + (\mathbf{S}^{-1} \cdot \mathbf{R}) \cdot (\mathbf{P} - \mathbf{Q} \cdot \mathbf{S}^{-1} \cdot \mathbf{R})^{-1} \cdot (\mathbf{Q} \cdot \mathbf{S}^{-1})$$

Tridiagonal systems:

Are ubiquitous in the solution of the partial differential equations of physics. Since these equations are second order, the discretized second derivatives normally involve evaluating the unknown at point $j, j-1, j+1$. If one views the solution evaluated at each point of the grid as a vector, then the discretized differential equation will look like a tridiagonal matrix times the vector (boundary conditions spoil this a bit, but not too much, we will discuss that immediately).

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & & \\ a_2 & b_2 & c_2 & \cdots & & \\ & & \cdots & & & \\ & & \cdots & a_{N-1} & b_{N-1} & c_{N-1} \\ & & \cdots & 0 & a_N & b_N \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ \cdots \\ u_{N-1} \\ u_N \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_{N-1} \\ r_N \end{bmatrix}$$

The matrix A is stored as three vectors a, b, c . It is worthwhile writing an elimination routine specifically, because the operations are very simple and compact.

```

subroutine tridag(a,b,c,r,u,n)
  parameter (nmax=100)
  dimension gam(nmax),a(n),b(n),c(n),r(n),u(n)
  if(b(1).eq.0.)pause
  bet=b(1)
  u(1)=r(1)/bet
  do 11 j=2,n
    gam(j)=c(j-1)/bet
    bet=b(j)-a(j)*gam(j)
    if(bet.eq.0.)pause
    u(j)=(r(j)-a(j)*u(j-1))/bet
11  continue
  do 12 j=n-1,1,-1
    u(j)=u(j)-gam(j+1)*u(j+1)
12  continue
  return
end

```

Decomposition and forward substitution $L \bullet y = b$

Backsubstitution $U \bullet y = x$

But what if your system is “almost” tri- or band-diagonal, by “almost” meaning there are some extra non-zero elements floating around? One could use the block technique, but many elements in the blocks will be zero. It is best to find an alternative technique, called the Sherman-Morrison formula.

Sherman-Morrison formula:

Suppose you have a great method for dealing with a sparse matrix \mathbf{A} , but you need to consider a matrix that differs from \mathbf{A} by terms of the form,

$$\mathbf{A} \rightarrow (\mathbf{A} + \mathbf{u} \otimes \mathbf{v})$$

Where \mathbf{u} and \mathbf{v} are vectors. For instance if \mathbf{u} and \mathbf{v} are unit vectors, they add one element to \mathbf{A} in the position i,j corresponding to their non-vanishing components. If \mathbf{u} is a unit vector and \mathbf{v} is generic, then one adds a column at that position.

$$\begin{aligned}(\mathbf{A} + \mathbf{u} \otimes \mathbf{v})^{-1} &= (\mathbf{1} + \mathbf{A}^{-1} \cdot \mathbf{u} \otimes \mathbf{v})^{-1} \cdot \mathbf{A}^{-1} \\&= (\mathbf{1} - \mathbf{A}^{-1} \cdot \mathbf{u} \otimes \mathbf{v} + \mathbf{A}^{-1} \cdot \mathbf{u} \otimes \mathbf{v} \cdot \mathbf{A}^{-1} \cdot \mathbf{u} \otimes \mathbf{v} - \dots) \cdot \mathbf{A}^{-1} \\&= \mathbf{A}^{-1} - \mathbf{A}^{-1} \cdot \mathbf{u} \otimes \mathbf{v} \cdot \mathbf{A}^{-1} (1 - \lambda + \lambda^2 - \dots) \\&= \mathbf{A}^{-1} - \frac{(\mathbf{A}^{-1} \cdot \mathbf{u}) \otimes (\mathbf{v} \cdot \mathbf{A}^{-1})}{1 + \lambda} \qquad \lambda \equiv \mathbf{v} \cdot \mathbf{A}^{-1} \cdot \mathbf{u}\end{aligned}$$

Or, defining, $\mathbf{z} \equiv \mathbf{A}^{-1} \cdot \mathbf{u}$ $\mathbf{w} \equiv (\mathbf{A}^{-1})^T \cdot \mathbf{v}$ $\lambda = \mathbf{v} \cdot \mathbf{z}$

$$\mathbf{A}^{-1} \rightarrow \mathbf{A}^{-1} - \frac{\mathbf{z} \otimes \mathbf{w}}{1 + \lambda}$$

A way to recast these results is to consider the two linear problems,

$$\mathbf{A} \cdot \mathbf{y} = \mathbf{b} \quad \mathbf{A} \cdot \mathbf{z} = \mathbf{u}$$

Then the vector \mathbf{x} , given by $\mathbf{x} = \mathbf{y} - \left[\frac{\mathbf{v} \cdot \mathbf{y}}{1 + (\mathbf{v} \cdot \mathbf{z})} \right] \mathbf{z}$

Is a solution (exercise) of the linear problem

$$(\mathbf{A} + \mathbf{u} \otimes \mathbf{v}) \cdot \mathbf{x} = \mathbf{b}$$

This approach is useful if storing \mathbf{A}^{-1} is prohibitive for reasons of space.

Cyclic tridiagonal systems are a good example of application of the Sherman-Morrison construction. They arise in solving PDE's with periodic boundary conditions,

$$\begin{bmatrix} b_1 & c_1 & 0 & \cdots & & & \beta \\ a_2 & b_2 & c_2 & \cdots & & & \\ & & \cdots & & & & \\ & & \cdots & a_{N-1} & b_{N-1} & c_{N-1} & \\ \alpha & & \cdots & 0 & a_N & b_N & \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_{N-1} \\ r_N \end{bmatrix}$$

This is an almost tridiagonal matrix with,

$$\mathbf{u} = \begin{bmatrix} \gamma \\ 0 \\ \vdots \\ 0 \\ \alpha \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ \beta/\gamma \end{bmatrix}$$

Where γ is arbitrary. It is good to chose it equal to $-b_1$ so it does not produce a cancellation in the first forward substitution.

```

SUBROUTINE cyclic(a,b,c,alpha,beta,r,x,n)
INTEGER n,NMAX
REAL alpha,beta,a(n),b(n),c(n),r(n),x(n)
PARAMETER (NMAX=500)
USES tridag
    Solves for a vector  $x(1:n)$  the "cyclic" set of linear equations given by equation (2.7.9).
    a, b, c, and r are input vectors, while alpha and beta are the corner entries in the matrix.
    The input is not modified.
INTEGER i
REAL fact,gamma,bb(NMAX),u(NMAX),z(NMAX)
if(n.le.2)pause 'n too small in cyclic'
if(n.gt.NMAX)pause 'NMAX too small in cyclic'
gamma=-b(1)                                Avoid subtraction error in forming bb(1).
bb(1)=b(1)-gamma                            Set up the diagonal of the modified tridiagonal system.
bb(n)=b(n)-alpha*beta/gamma
do 11 i=2,n-1
    bb(i)=b(i)
enddo 11
call tridag(a,bb,c,r,x,n)                  Solve  $A \cdot x = r$ .
u(1)=gamma                                Set up the vector  $u$ .
u(n)=alpha
do 12 i=2,n-1
    u(i)=0.
enddo 12
call tridag(a,bb,c,u,z,n)                  Solve  $A \cdot z = u$ .
fact=(x(1)+beta*x(n)/gamma)/(1.+z(1)+beta*z(n)/gamma)    Form  $v \cdot x / (1 + v \cdot z)$ .
do 13 i=1,n
    x(i)=x(i)-fact*z(i)                    Now get the solution vector  $x$ .
enddo 13
return
END

```

There are many tricks for indexing efficiently sparse matrices. Most are not terribly enlightening and require careful study, so we will not cover them in class. Numerical recipes lists a few.

Conjugate gradient method:

If you have a square matrix that is positive definite (its eigenvalues are all positive), or $\mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x} > 0, \quad \forall \mathbf{x}$ you can construct the function,

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x} - \mathbf{b} \cdot \mathbf{x}$$

And the condition for this function to have a minimum is that its gradient vanishes, that is,

$$\nabla f = \mathbf{A} \cdot \mathbf{x} - \mathbf{b}$$

Therefore the \mathbf{x} that minimizes the function solves the linear system of equations. The technique can be (non-trivially) generalized to non-positive definite matrices.

We will discuss function minimization later in the course.

Summary

- Approximate methods can lead to less problems than exact ones.
- Sparse matrices are treated in blocks.
- Sherman-Morrison for “almost tridiagonal”
- Conjugate gradient takes advantages of minimization techniques that can converge very fast.