Lecture 11

Last time talked about Gaussian elimination, now discuss the LU decomposition ...

Often we want to solve equations of the form $Ax = v$ for the same A but different vectors v.

We would like to do this in a way such that we don't have to solve Gaussian elimination every time.

For this we can use the LU decomposition.

Let $A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{03} \\ a_{10} & a_{11} & & \\ \vdots & & \ddots & \\ a_{30} & \cdots & & a_{33} \end{bmatrix}$

In Gaussian elimination,

we first divide out first row by a_{00}. Then use the entry there to cancel out ~~other~~ 1st entries on other rows.

This can be written as a matrix

$$\frac{1}{a_{00}}\begin{bmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{bmatrix} = L_0$$

Left Multiplying original matrix by this gives

$$\begin{bmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Observe that this is lower triangular

Next step is to divide out second row by b_{11} & subtract from lower rows, can write as a matrix also

$$\frac{1}{b_{11}} \begin{bmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{bmatrix} = L_1$$

Then L_1 times previous matrix gives

$$\begin{bmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{bmatrix}$$

can continue this to get two more matrices L_2 & L_3

Then we have

$$\underbrace{L_3 L_2 L_1 L_0 A}_{\text{this is an upper triangular matrix}} x = \overbrace{L_3 L_2 L_1 L_0 V}^{\text{all known}}$$

can then use back-substitution to solve.

Advantage is that we can use

$L_3 \cdots L_0$ on any vectors v.

In practice, things are done slightly differently.

Let $L = L_0^{-1} L_1^{-1} L_2^{-1} L_3^{-1}$

$$U = L_3 L_2 L_1 L_0 A$$

$$\Rightarrow \quad A = LU$$

So that $\quad LUx = v$

$Ax = v$

L actually has a simple form

Consider $L_0 = \dfrac{1}{a_{00}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{10} \end{bmatrix}$

5

Then $L_0^{-1} = \begin{bmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & 1 & 0 & 0 \\ a_{20} & 0 & 1 & 0 \\ a_{30} & 0 & 0 & 1 \end{bmatrix}$

Similar form for $L_1^{-1}, L_2^{-1}, L_3^{-1}$ so that

$$L = L_0^{-1} L_1^{-1} L_2^{-1} L_3^{-1} = \begin{bmatrix} a_{00} & 0 & 0 & 0 \\ a_{10} & b_{11} & 0 & 0 \\ a_{20} & b_{21} & c_{22} & 0 \\ a_{30} & b_{31} & c_{32} & d_{33} \end{bmatrix}$$

Once we have an LU decomposition, we can solve $Ax = v$ directly as

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} l_{00} & 0 & 0 \\ l_{10} & l_{11} & 0 \\ l_{20} & l_{21} & l_{22} \end{bmatrix} \cdot \begin{bmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{bmatrix}$$

$A = LU$

$\Rightarrow Ax = LUx = v$

define y as $y = Ux$

then

$Ly = v$

can back substitute here, figure out y & then use $Ux = y$ & back substitution to get x.

To avoid issues w/ small or zero elements, use pivoting or partial pivoting...

So process breaks down into finding LU decomposition & then 2 back-subst.

To solve systems of linear equations in Python, you can just use

```
from numpy.linalg import solve
x = solve(A, v)
```

Calculating a matrix inverse

can use what we already know to solve systems of the form

$$AX = V$$

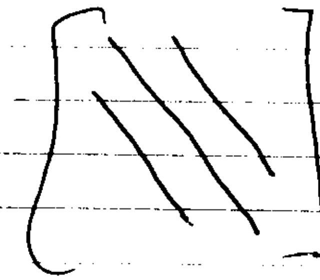
for matrix X + matrix V.

do each column separately

Then if we set $V = I$ we are solving for inverse of A.

Tridiagonal & banded matrices

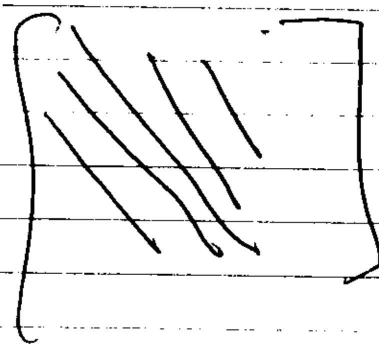
Consider $A =$



Gaussian elimination works well but no need to do it in full.

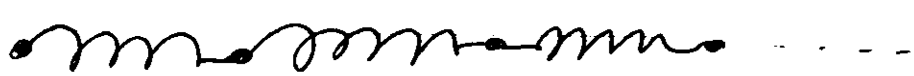
Just subtract each row from the one immediately below it.

Similar kind of procedure for banded matrices



Example:

given are N identical masses connected by identical springs



Let ξ_i denote the displacement of i^{th} mass relative to rest position

Then the Newton 2^{nd} law is that

$$m\frac{d^2\xi_i}{dt^2} = k(\xi_{i+1}-\xi_i) + k(\xi_{i-1}-\xi_i) + F_i$$

where m is mass, k is spring constant + F_i is any external force on mass i. Endpoint masses are described by

$$m\frac{d^2\xi_1}{dt^2} = k(\xi_2-\xi_1) + F_1$$

& $$m\frac{d^2\xi_N}{dt^2} = k(\xi_{N-1} - \xi_N) + F_N$$

Now drive the system w/ a periodic force applied to 1st mass

$F_1 = Ce^{i\omega t}$ & C is a constant

(take real part at the end)

net result of applied force will be to make masses oscillate w/ angular frequency ω so that

$\xi_i(t) = x_i e^{i\omega t}$

can substitute to find

$$-m\omega^2 x_1 = k(x_2 - x_1) + C$$

$$-m\omega^2 x_i = k(x_{i+1} - x_i) + k(x_{i-1} - x_i)$$

$$-m\omega^2 x_N = k(x_{N-1} - x_N)$$

can rearrange as

$$(\alpha - k)\, x_1 - k x_2 = C$$

$$\alpha x_i - k x_{i-1} - k x_{i+1} = 0$$

$$(\alpha - k)\, x_N - k x_{N-1} = 0$$

where $\alpha = 2k - m\omega^2$

$\Rightarrow$ matrix form

$$\begin{bmatrix} \alpha - k & -k & & & & \\ -k & \alpha & -k & & & \\ & -k & \alpha & -k & & \\ & & & \ddots & & \\ & & & -k & \alpha & -k \\ & & & & \alpha & \alpha - k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} C \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Bring up 1-springs.py

plot demonstrate that some move vigorously while others do not

Eigenvalues + eigenvectors

# of important applications in physics

Let A be a symmetric matrix.

Eigenvector v is such that

$$Av = \lambda v$$

for scalar λ.

$N \times N$ matrix has N eigenvectors w/ N eigenvalues. Eigenvectors can be taken orthonormal.

can take eigenvectors $\{v_i\}$ to be columns of single matrix V + write all equations as $Av_i = \lambda_i v_i$ as a single matrix equation

$$AV = V\Lambda$$

where Λ is diagonal matrix of eigenvalues. V is an orthogonal matrix, so that $V^T V = V V^T = I$

Most widely used technique for getting eigenvectors & eigenvalues is QR algorithm.

How does it work?

First calculate the QR decomposition of a matrix A as

$$A = QR$$

where Q is orthogonal & R is upper triangular. Any square matrix A can be written like this.

So the 1st step of algorithm is

$$A = Q_1 R_1$$

Multiply by Q_1^T on left to get

$$Q_1^T A = Q_1^T Q_1 R_1 = R_1$$

Now let $A_1 = R_1 Q_1$ which is the reverse of A_1. Then

$$A_1 = R_1 Q_1 = Q_1^T A Q_1$$

Then we iterate this process:

find QR decomposition of A_1 as $A_1 = Q_2 R_2$

define

$$A_2 = R_2 Q_2 = Q_2^T A_1 Q_2$$
$$= Q_2^T Q_1^T A Q_1 Q_2$$

continuing, we get

$$A_3 = Q_3^T Q_2^T Q_1^T A Q_1 Q_2 Q_3$$

$$\vdots$$

$$A_k = (Q_k^T \cdots Q_1^T) A (Q_1 \cdots Q_k)$$

This procedure converges such that A_k is diagonal. Convergence rate is given by

$$[A_k]_{i,j} = O\left(\left|\frac{\lambda_i}{\lambda_j}\right|^k\right) \text{ for } i > j$$

& where eigenvalues are sorted as

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > \dots > |\lambda_n| > 0$$

Furthermore,

$$\lim_{k\to\infty} Q_k = I \,, \quad \lim_{k\to\infty} A_k = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$$

Define $V = Q_1 \cdots Q_k$

Then $\Lambda = V^T A V$

& $AV = V\Lambda$ so that

V is matrix of eigenvectors &

Λ is diagonal matrix of eigenvalues

Complete QR algorithm is then:

Given $N \times N$ matrix A

1. Create $N \times N$ V & set $V = I$. Choose $\varepsilon > 0$ as desired accuracy.
2. Calculate QR decomp. as $A = QR$
3. Update A to $A = RQ$

4. Set $V = VQ$

5. Check magnitude of all off-diagonal elements of A. If all are $\leq \varepsilon$, then stop. Otherwise, go to step 2.

There are a variety of improvements to this, which we won't discuss.

Description of QR algorithm

Think of A as $\begin{bmatrix} | & | & | & \\ a_0 & a_1 & a_2 & \cdots \\ | & | & | & \end{bmatrix}$

Let $u_0 = a_0 \qquad q_0 = \dfrac{u_0}{\|u_0\|}$

$u_1 = a_1 - (q_0 \cdot a_1) q_0 \qquad q_1 = \dfrac{u_1}{\|u_1\|}$

$u_2 = a_2 - (q_0 \cdot a_2) q_0 - (q_1 \cdot a_2) q_1$

$q_2 = \dfrac{u_2}{\|u_2\|}$

$\vdots$

(each time subtracting out ~~components~~ projection of a_i onto orthonormal subspace)

general formulas are

$$u_i = a_i - \sum_{j=0}^{i-1} (q_j \cdot a_i)\, q_j$$

$$q_i = \frac{u_i}{\|u_i\|}$$

can show that $\{q_i\}$ is an O.N. basis

Then

$$a_0 = \|u_0\| q_0$$

$$a_1 = \|u_1\| q_1 + (q_0 \cdot a_1) q_0$$

$$a_2 = \|u_2\| q_2 + (q_0 \cdot a_2) q_0 + (q_1 \cdot a_2) q_1$$

can write these as

$$A = \begin{bmatrix} | & | & | & \\ a_0 & a_1 & a_2 & \cdots \\ | & | & | & \end{bmatrix} = \begin{bmatrix} | & | & | & \\ q_0 & q_1 & q_2 & \cdots \\ | & | & | & \end{bmatrix} \times$$

$$\begin{bmatrix} \|u_0\| & q_0 \cdot a_1 & q_0 \cdot a_2 & \cdots \\ & \|u_1\| & q_1 \cdot a_2 & \cdots \\ 0 & & & \\ 0 & 0 & \|u_2\| & \ddots \\ \vdots & & \ddots & \ddots \end{bmatrix}$$

so left matrix is orthogonal

& right one is upper triangular