Chap. 6
Solution of Linear and Nonlinear Equations

Soon-Hyung Yook

May 29, 2017
Table of Contents I

1 Simultaneous Linear Equations
   - Gauss-Jordan Elimination
   - Backsubstitution
   - Pivoting
   - LU Decomposition
   - Inverse of a Matrix
   - Tridiagonal and Banded Matrices

2 Eigenvalues and Eigenvectors
Simultaneous Linear Equations I

- Simultaneous set of linear equation:
  - One can solve the equation by using paper and a pen!
  - But if there are many variables, then the procedure is very tedious.
  - Moreover, humans are slow and prone to error in such tedious calculations.

Example: four simultaneous equations with four variables, \( w, x, y \) and \( z \).

\[
\begin{align*}
2w + x + 4 + z &= -4, \\
3w + 4x - y - z &= 3, \\
w - 4x + y + 5z &= 9, \\
2w - 2x + y + 3z &= 7
\end{align*}
\] (1)

In matrix form

\[
\begin{pmatrix}
2 & 1 & 4 & 1 \\
3 & 4 & -1 & -1 \\
1 & -4 & 1 & 5 \\
2 & -2 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
=
\begin{pmatrix}
-4 \\
3 \\
9 \\
7
\end{pmatrix}.
\] (2)
Simultaneous Linear Equations

Simultaneous Linear Equations II

Alternatively, in a shorthand form:

\[ Ax = v, \]  

(3)

where \( x = (w, x, y, z) \) and the matrix \( A \) and vector \( v \) take the appropriate values. Then find the solution:

\[ x = A^{-1}v. \]  

(4)

But the problem is finding \( A^{-1} \) is not so trivial using computer.
Gauss-Jordan Elimination

- The most straightforward method to find the solution of Eq. (3).
- Two rules for Gauss-Jordan elimination:
  1. If we multiply any row of the matrix $A$ by any constant, and we multiply the corresponding row of the vector $v$ by the same constant, then the solution does not change.
  2. If we add to or subtract from any row of $A$ a multiple of any other row, and we do the same for the vector $v$, then the solution does not change.
As for an example, let’s try to solve Eq. (2) by hand:

1. Divide the first row by the top-left element of the matrix:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
3 & 4 & -1 & -1 \\
1 & -4 & 1 & 5 \\
2 & -2 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= \begin{pmatrix}
-2 \\
3 \\
9 \\
7
\end{pmatrix}.
\] (5)

2. Subtract 3 times the first row from the second row:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 2.5 & -7 & -2.5 \\
1 & -4 & 1 & 5 \\
2 & -2 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= \begin{pmatrix}
-2 \\
9 \\
9 \\
7
\end{pmatrix}.
\] (6)
Example: Eq. (2) II

3 Subtract the first row from the third one, and also subtract 2 times the first row from the fourth:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 2.5 & -7 & -2.5 \\
0 & -4.5 & -1 & 4.5 \\
0 & -3 & -3 & 2
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= 
\begin{pmatrix}
-2 \\
9 \\
11 \\
11
\end{pmatrix}.
\] (7)

4 Divide the second row by 2.5:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 1 & -2.8 & -1 \\
0 & -4.5 & -1 & 4.5 \\
0 & -3 & -3 & 2
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= 
\begin{pmatrix}
-2 \\
3.6 \\
11 \\
11
\end{pmatrix}.
\] (8)
Example: Eq. (2) III

5 Subtract $-4.5$ times the second row from the third, and $-3$ times the second row from the fourth:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 1 & -2.8 & -1 \\
0 & 0 & -13.6 & 0 \\
0 & 0 & -11.4 & -1
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= \begin{pmatrix}
-2 \\
3.6 \\
27.2 \\
21.8
\end{pmatrix}.
\]  \hspace{1cm} (9)

6 Divide the third row by $-13.6$:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 1 & -2.8 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & -11.4 & -1
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= \begin{pmatrix}
-2 \\
3.6 \\
-2 \\
21.8
\end{pmatrix}.
\]  \hspace{1cm} (10)
Example: Eq. (2) IV

7 Subtract \(-11.4\) times third row from the fourth:

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 1 & -2.8 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= 
\begin{pmatrix}
-2 \\
3.6 \\
-2 \\
-1
\end{pmatrix}. 
\tag{11}
\]

8 Divide the fourth row by \(-1\):

\[
\begin{pmatrix}
1 & 0.5 & 2 & 0.5 \\
0 & 1 & -2.8 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
= 
\begin{pmatrix}
-2 \\
3.6 \\
-2 \\
1
\end{pmatrix}. 
\tag{12}
\]

By definition, Eq. (12) has the same solution with Eq. (2), but the matrix is now upper triangular.
Backsubstitution I

- To find the final solution of Eq. (2) we now use the process of backsubstitution.
- Suppose we have any set of equations of the form:

\[
\begin{pmatrix}
1 & a_{01} & a_{02} & a_{03} \\
0 & 1 & a_{12} & a_{13} \\
0 & 0 & 1 & a_{23} \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix} =
\begin{pmatrix}
v_0 \\
v_1 \\
v_2 \\
v_3
\end{pmatrix}.
\] (13)

- Eq. (13) can be written as:

\[
w + a_{01}x + a_{02}y + a_{03}z = v_0, 
\] (14)

\[
x + a_{12}y + a_{13}z = v_1, 
\] (15)

\[
y + a_{23}z = v_2, 
\] (16)

\[
z = v_3. 
\] (17)
Backsubstitution II

1. From Eq. (17):
   \[ z = v_3 \]  

2. From Eq. (16)
   \[ y = v_2 - a_{23}z \]  

3. From Eq. (15)
   \[ x = v_1 - a_{12}y - a_{13}z \]  

4. From Eq. (14)
   \[ w = v_0 - a_{01}x - a_{02}y - a_{03}z \]  

Applying Eqs. (18)-(21) we obtain:

   \[ w = 2, \quad x = -1, \quad y = -2, \quad z = 1. \]  

(22)
Example 6.1: I

Guassian elimination for Eq. (2):

```python
from numpy import array, empty

A=array([[2,1,4,1],
          [3,4,-1,-1],
          [1,-4,1,5],
          [2,-2,1,3]], float)

v=array([-4,3,9,7], float)
N=len(v)

# Gaussian Elimination
for m in range(N):
    # Divide by the diagonal element
    div=A[m,m]
    A[m,:]/=div
    v[m]/=div

    # Subtract from the lower rows
    for i in range(m+1,N):
        mult=A[i,m]
        A[i,:]-=mult*A[m,:]
        v[i]-=mult*v[m]

# Backsubstitution
```
Example 6.1: II

```
x=empty(N, float)
for m in range(N-1, -1, -1):
    x[m]=v[m]
    for i in range(m+1,N):
        x[m]=A[m, i]*x[i]

print(x)
```
Now let’s consider the equations:

\[
\begin{pmatrix}
0 & 1 & 4 & 1 \\
3 & 4 & -1 & -1 \\
1 & -4 & 1 & 5 \\
2 & -2 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix}
=
\begin{pmatrix}
-4 \\
3 \\
9 \\
7
\end{pmatrix}.
\]  

- Here the first element of the first row is zero!
- This causes a problem to apply Gauss-Jordan elimination.
  - Divide by zero is not allowed.
Pivoting II

Pivoting

Exchange rows:

\[
\begin{pmatrix}
3 & 4 & -1 & -1 \\
0 & 1 & 4 & 1 \\
1 & -4 & 1 & 5 \\
2 & -2 & 1 & 3
\end{pmatrix}
\begin{pmatrix}
w \\
x \\
y \\
z
\end{pmatrix} =
\begin{pmatrix}
3 \\
-4 \\
9 \\
7
\end{pmatrix}.
\]  

(24)

Partial Pivoting

With partial pivoting, we consider rearranging the rows at each stage.

- When we get to the \( m \)th row, we compare it to all lower rows, looking at the value each row has in its \( m \)th elements and finding the one such value that is the farthest from zero—either positive or negative.
- If the row containing this winning value is not currently \( m \)th row, then we move it up to \( m \)th place by swapping it with the current \( m \)th row.
Example–Exercise 6.2 I

Solve Eq. (23) using Gauss-Jordan elimination with partial pivoting.

```python
from numpy import array, empty

A = array([[0, 1, 4, 1],
           [3, 4, -1, -1],
           [1, -4, 1, 5],
           [2, -2, 1, 3]], float)

v = array([-4, 3, 9, 7], float)
N = len(v)

# Gaussian Elimination
for m in range(N):
    # Applying partial pivoting
    pivot_max = abs(A[m, m])
    pivot_point = m
    for i in range(m + 1, N):
        pivot_tmp = abs(A[i, m])
        if pivot_tmp > pivot_max:
            pivot_point, pivot_max = i, pivot_tmp
    if m != pivot_point:
        for i in range(N):
            A[m, i], A[pivot_point, i] = A[pivot_point, i], A[m, i]
        v[m], v[pivot_point] = v[pivot_point], v[m]
```

Soon-Hyung Yook
Chap. 6
May 29, 2017 16 / 62
Simultaneous Linear Equations

Example—Exercise 6.2 II

```python
print(A)
input()
# Divide by the diagonal element
div=A[m,m]
A[m,:]/=div
v[m]/=div

# Subtract fro the lower rows
for i in range(m+1,N):
    mult=A[i,m]
    A[i,:]=mult*A[m,:]
    v[i]=mult*v[m]

# Backsubtraction
x=empty(N, float)
for m in range(N-1,-1,-1):
    x[m]=v[m]
    for i in range(m+1,N):
        x[m]=A[m,i]*x[i]
print(x)
```
Gauss-Jordan Elimination in Matrix Form I

- Basically based on the Gauss-Jordan elimination method.
- Powerful when we have to solve many different sets of equations $Ax = v$ with the same matrix $A$ but different right-hand sides $v$.
  - Repeating Gauss-Jordan elimination would be time-consuming.

Suppose we have a $4 \times 4$ matrix

$$A = \begin{pmatrix}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
a_{30} & a_{31} & a_{32} & a_{33}
\end{pmatrix} \quad (25)$$

The Gauss-Jordan elimination is written as a matrix form:
Step 1:

$$\frac{1}{a_{00}} \begin{pmatrix}
1 & 0 & 0 & 0 \\
-a_{10} & a_{00} & 0 & 0 \\
-a_{20} & 0 & a_{00} & 0 \\
-a_{30} & 0 & 0 & a_{00}
\end{pmatrix} \begin{pmatrix}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
a_{30} & a_{31} & a_{32} & a_{33}
\end{pmatrix} = \begin{pmatrix}
1 & b_{01} & b_{02} & b_{03} \\
0 & b_{11} & b_{12} & b_{13} \\
0 & b_{21} & b_{22} & b_{23} \\
0 & b_{31} & b_{32} & b_{33}
\end{pmatrix} \quad (26)$$
Define a *lower triangular* matrix $L_0$ as

$$L_0 = \frac{1}{a_{00}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{pmatrix} \tag{27}$$

Step 2:

$$\frac{1}{b_{11}} \begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix} \begin{pmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{pmatrix} \tag{28}$$

Define another lower triangular matrix $L_1$ as

$$L_1 = \frac{1}{b_{11}} \begin{pmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{pmatrix} \tag{29}$$
Gauss-Jordan Elimination in Matrix Form III

Step 3:

\[
\frac{1}{c_{22}} \begin{pmatrix} c_{22} & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{32} & c_{22} \end{pmatrix} \begin{pmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{pmatrix} = \begin{pmatrix} 1 & d_{01} & d_{02} & d_{03} \\ 0 & 1 & d_{12} & d_{13} \\ 0 & 0 & 1 & d_{23} \\ 0 & 0 & 0 & d_{33} \end{pmatrix} \tag{30}
\]

And define \( L_2 \) as

\[
L_2 = \frac{1}{c_{22}} \begin{pmatrix} c_{22} & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{32} & c_{22} \end{pmatrix}
\tag{31}

Step 4:

\[
\frac{1}{d_{33}} \begin{pmatrix} d_{33} & 0 & 0 & 0 \\ 0 & d_{33} & 0 & 0 \\ 0 & 0 & d_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & d_{01} & d_{02} & d_{03} \\ 0 & 1 & d_{12} & d_{13} \\ 0 & 0 & 1 & d_{23} \\ 0 & 0 & 0 & d_{33} \end{pmatrix} = \begin{pmatrix} 1 & u_{01} & u_{02} & u_{03} \\ 0 & 1 & u_{12} & u_{13} \\ 0 & 0 & 1 & u_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{32}
\]
Gauss-Jordan Elimination in Matrix Form IV

Define $L_3$ as

$$L_3 = \frac{1}{d_{33}} \begin{pmatrix} d_{33} & 0 & 0 & 0 \\ 0 & d_{33} & 0 & 0 \\ 0 & 0 & d_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(33)

Step 1-Step 4 are mathematically written as

$$L_3L_2L_1L_0A$$

Therefore, we solve our original set of equations $Ax = v$ by multiplying $L_3L_2L_1L_0$ as

$$L_3L_2L_1L_0Ax = L_3L_2L_1L_0v$$

(34)

Then apply the backsubstitution.
In practice, we don’t need to have all four matrix \( L_0 \), \( L_1 \), \( L_2 \), and \( L_3 \).

- Define two matrices:
  \[
  L = L_0^{-1}L_1^{-1}L_2^{-1}L_3^{-1}, \quad \quad \quad \quad \quad \quad U = L_3L_2L_1L_0A \tag{35}
  \]

- Note that \( U \) is the upper triangular matrix (right-hand side of Eq. (32)).
- Multiplying \( L \) and \( U \) gives
  \[
  LU = A \tag{36}
  \]
- Form the original set of equations, \( Ax = v \),
  \[
  LUx = v \tag{37}
  \]
- Note that \( L \) is lower triangular matrix.
Consider the matrix $L_0$ for example:

$$L_0 = \frac{1}{a_{00}} \begin{pmatrix}
1 & 0 & 0 & 0 \\
-a_{10} & a_{00} & 0 & 0 \\
-a_{20} & 0 & a_{00} & 0 \\
-a_{30} & 0 & 0 & a_{00}
\end{pmatrix} \quad (38)$$

Inverse of $L_0$ is

$$L_0^{-1} = \begin{pmatrix}
a_{00} & 0 & 0 & 0 \\
a_{10} & 1 & 0 & 0 \\
a_{20} & 0 & 1 & 0 \\
a_{30} & 0 & 0 & 1
\end{pmatrix} \quad (39)$$

It can be easily verified by showing $L_0L_0^{-1} = I$, where $I$ is an identity matrix (or more precisely see Boas’s book).
Simultaneous Linear Equations
LU Decomposition

LU Decomposition III

- Similarly,
  \[
  L^{-1}_1 = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & b_{11} & 0 & 0 \\
  0 & b_{21} & 1 & 0 \\
  0 & b_{31} & 0 & 1 \\
  \end{pmatrix} \quad L^{-1}_2 = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & c_{22} & 0 \\
  0 & 0 & c_{32} & 1 \\
  \end{pmatrix} \quad L^{-1}_c = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & d_{33} \\
  \end{pmatrix}
  \]

- Multiplying them all together:

\[
L = L^{-1}_0 L^{-1}_1 L^{-1}_2 L^{-1}_3 = \begin{pmatrix}
  a_{00} & 0 & 0 & 0 \\
  a_{10} & b_{11} & 0 & 0 \\
  a_{20} & b_{21} & c_{22} & 0 \\
  a_{30} & b_{31} & c_{32} & d_{33} \\
  \end{pmatrix}
\] (41)

- Not only is \( L \) is lower triangular, but its elements are easily obtained through Gauss-Jordan elimination.
LU Decomposition-Backsubtraction I

To find a rule for backsubstitution, let’s consider a $3 \times 3$ matrix $A$.

- The LU decomposition of $A$ looks like:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{00} & 0 & 0 \\ l_{10} & l_{11} & 0 \\ l_{20} & l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{pmatrix}. \quad (42)$$

- Then the linear equations $Ax = v$ becomes

$$\begin{pmatrix} l_{00} & 0 & 0 \\ l_{10} & l_{11} & 0 \\ l_{20} & l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix}. \quad (43)$$

- Define a new vector $y$ as

$$\begin{pmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}. \quad (44)$$
Then Eq. (43) becomes
\[
\begin{pmatrix}
  l_{00} & 0 & 0 \\
  l_{10} & l_{11} & 0 \\
  l_{20} & l_{21} & l_{22}
\end{pmatrix}
\begin{pmatrix}
y_0 \\
y_1 \\
y_2
\end{pmatrix}
=
\begin{pmatrix}
v_0 \\
v_1 \\
v_2
\end{pmatrix}.
\tag{45}
\]

From the first line of Eq. (45), \( l_{00}y_0 = v_0 \). Thus
\[
y_0 = \frac{v_0}{l_{00}}.
\tag{46}
\]

From the second line of Eq. (45), \( l_{10}y_0 + l_{11}y_1 = v_1 \), or
\[
y_1 = \frac{v_1 - l_{10}y_0}{l_{11}}.
\tag{47}
\]

From the third line of Eq. (45) gives
\[
y_2 = \frac{v_2 - l_{20}y_0 - l_{21}y_1}{l_{22}}.
\tag{48}
\]
General Representation of $y$

$$y_i = \frac{v_i - \sum_{j=0}^{i-1} l_{ij} y_j}{l_{ii}}.$$  \hspace{1cm} (49)

- Applying partial pivoting is also trivial.

However, the simplest way to implement LU decomposition and backsubstitution is to use the `solve` function in `numpy.linalg` package like this:

```python
from numpy.linalg import solve
x = solve(A, v)
```
LU Decomposition: Example 1

Solve Eq. (23) using LU decomposition with partial pivoting.

```python
from numpy import array, zeros, empty, copy, dot
from numpy.linalg import solve

A = array([[0, 1, 4, 1],
           [3, 4, -1, -1],
           [1, -4, 1, 5],
           [2, -2, 1, 3]], float)
v = array([-4, 3, 9, 7], float)
N = len(v)
L = zeros([N, N], float)
U = empty([N, N], float)
U = copy(A)
print("A=", A)
print("U=", U)

# Gaussian Elimination with LU decomposition
for m in range(N):
    # Applying partial pivoting
    pivot_max = abs(U[m, m])
    pivot_point = m
    for i in range(m + 1, N):
        pivot_tmp = abs(U[i, m])
        if pivot_tmp > pivot_max:
            pivot_max = pivot_tmp
            pivot_point = i
    U[pivot_point, :] = U[m, :]
    U[:, pivot_point] = U[:, m]
```
LU Decomposition: Example II

```python
pivot_point, pivot_max = i, pivot_tmp
if m != pivot_point:
    for i in range(N):
        U[m, i], U[pivot_point, i] = U[pivot_point, i], U[m, i]
        L[m, i], L[pivot_point, i] = L[pivot_point, i], L[m, i]
        A[m, i], A[pivot_point, i] = A[pivot_point, i], A[m, i]
        v[m], v[pivot_point] = v[pivot_point], v[m]

    L[m:, m] = U[m:, m]

    # Divide by the diagonal element
    div = U[m, m]
    U[m, :] /= div

    # Subtract from the lower rows
    for i in range(m+1, N):
        mult = U[i, m]
        U[i, :] -= mult * U[m, :]

print()
print("After GE with LUD")
print("U=", U)
print()
print("L=", L)
print()
```
LU Decomposition: Example III

```python
print("A=", A)
print()
print("LU=", dot(L, U))

# Backsubtraction
y = empty(N, float)
for m in range(N):
    y[m] = v[m]
    for i in range(m):
        y[m] -= L[m, i] * y[i]
    y[m] /= L[m, m]

x = empty(N, float)
for m in range(N-1, -1, -1):
    x[m] = y[m]
    for i in range(m+1, N):
        x[m] -= U[m, i] * x[i]
    x[m] /= U[m, m]
print("\n")
print("x=", x)
print("solve(A, v)=", solve(A, v))
```
Calculating the Inverse of a Matrix I

Inverse of matrix:

\[ A^{-1} = \frac{1}{\det A} C^T \quad (50) \]

where \( C_{ij} \) is cofactor of \( a_{ij} \) (see the mathematical physics textbook).

- But calculating the determinants are time consuming and prone to make large error.

- Apply the method to solve simultaneous linear equations.

- Consider a form

\[ AX = V. \quad (51) \]

- Now, \( X \) and \( V \) are \( N \times N \) matrix as well as \( A \).

- If \( V = I \), then \( X \) is the inverse matrix of \( A \).
Calculating the Inverse of a Matrix II

Calculating the Inverse of a Matrix

Now we have to solve a set of \( N \) simultaneous linear equations:

\[
AX_j = V_j, \tag{52}
\]

where \( j = 0, 1, \ldots, N - 1 \).

- \( X_j \) is the \( j \)th column of matrix \( X \).
- \( V_j \) is the \( j \)th column of matrix \( V \).
- We set \( V = I \).

Then we can apply the Gauss-Jordan elimination or LU decomposition method for each column vector \( X_j \) and \( V_j \).

By combining \( X_j \)'s we can obtain \( X = A^{-1} \).

Of course we can also use \texttt{inv} function in \texttt{numpy.linalg} package as:

```python
from numpy.linalg import inv
X=inv(A)
```
Inverse Matrix: Example I

Find $A^{-1}$ in Eq. (23) using LU decomposition with partial pivoting.

```python
from numpy import array, zeros, empty, copy, dot
from numpy.linalg import inv

A = array([[0, 1, 4, 1],
           [3, 4, -1, -1],
           [1, -4, 1, 5],
           [2, -2, 1, 3]], float)

n = A.shape
N = n[1]
L = zeros([N, N], float)
U = empty([N, N], float)
U = copy(A)
V = zeros([N, N], float)
for m in range(N):
    V[m, m] = 1.0

print("A=", A)
predict ("U=", U)
predict ("V=", V)
predict ("inv(A)=", inv(A))    # for comparison

# Gaussian Elimination with LU decomposition
for m in range(N):
    # Applying partial pivoting
```
Inverse Matrix: Example II

```python
pivot_max = abs(U[m,m])
pivot_point = m
for i in range(m+1, N):
    pivot_tmp = abs(U[i,m])
    if pivot_tmp > pivot_max:
        pivot_point, pivot_max = i, pivot_tmp
if m != pivot_point:
    for i in range(N):
        U[m,i], U[pivot_point, i] = U[pivot_point, i], U[m, i]
        L[m,i], L[pivot_point, i] = L[pivot_point, i], L[m, i]
        A[m,i], A[pivot_point, i] = A[pivot_point, i], A[m, i]
        V[m,i], V[pivot_point, i] = V[pivot_point, i], V[m, i]
L[m:,m] = U[m:,m]
# Divide by the diagonal element
div = U[m,m]
U[m,:] /= div

# Subtract from the lower rows
for i in range(m+1, N):
    mult = U[i,m]
    U[i,:] -= mult * U[m,:]
# Now we have L and U
```
Inverse Matrix: Example III

```python
Y=empty([N,N], float )
for j in range(N):  # for each column
    for m in range(N):  # for each row
        Y[m, j]=V[m, j ]
        for i in range(m):
            Y[m, j]−=L[m, i ]*Y[i , j ]
        Y[m, j] /=L[m,m]

X=empty([N,N], float )
for j in range(N):
    for m in range(N−1,−1,−1):
        X[m, j]=Y[m, j ]
        for i in range(m+1,N):
            X[m, j]−=U[m, i ]*X[i , j ]
        X[m, j] /=U[m,m]

print("\n")
print("X=" ,X)
```
Tridiagonal Matrices: Trigonal Matrix Algorithm or Thomas Algorithm

A special case that arise often in physics problems is the solution of $Ax = v$ when the matrix $A$ is tridiagonal:

$$A = \begin{pmatrix}
a_{00} & a_{01} & 0 & 0 & 0 \\
a_{10} & a_{11} & a_{12} & 0 & 0 \\
0 & a_{21} & a_{22} & a_{23} & 0 \\
0 & 0 & a_{32} & a_{33} & a_{34} \\
0 & 0 & 0 & a_{43} & a_{44}
\end{pmatrix}.$$

(53)

- The matrix has nonzero elements only along the diagonal and immediately above and below it.
- Simple Gauss-Jordan elimination is a good choice for solving the problem.
  - Quick
  - Pivoting is typically not used
    - Thus, the programming is straightforward.
  - We do not need to go through the entire Gauss-Jordan elimination process.
    - Each row only need to be subtracted from the single row immediately below it – and not all lower rows – to make the matrix triangular.
Consider a $4 \times 4$ matrix:

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 4 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \quad (54)$$

1. **Step 1**: Divide the first row by 2, then subtract 3 times the result from the second row:

$$\begin{pmatrix} 1 & 0.5 & 0 & 0 \\ 0 & 2.5 & -5 & 0 \\ 0 & -4 & 3 & 5 \\ 0 & 0 & 1 & 3 \end{pmatrix} \quad (55)$$
Illustration: How to Make the Matrix Triangular II

Step 2: Divide the second row by 2.5 and subtract $-4$ times the result from the third row:

$$
\begin{pmatrix}
1 & 0.5 & 0 & 0 \\
0 & 1 & -2 & 0 \\
0 & 0 & -5 & 5 \\
0 & 0 & 1 & 3 \\
\end{pmatrix}
$$

(56)

Step 3: Divide the third row by $-5$ and subtract it from the fourth row:

$$
\begin{pmatrix}
1 & 0.5 & 0 & 0 \\
0 & 1 & -2 & 0 \\
0 & 0 & 1 & -1 \\
0 & 0 & 0 & 4 \\
\end{pmatrix}
$$

(57)
Illustration: How to Make the Matrix Triangular III

Step 4: Divide the fourth row by 4, then we obtain upper triangular matrix:

\[
\begin{pmatrix}
1 & 0.5 & 0 & 0 \\
0 & 1 & -2 & 0 \\
0 & 0 & 1 & -1 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]  \tag{58}

- Note that green colored elements are not changed when subtracting some constant multiple of the above row.
- Use this fact to reduce the computing time.
Illustration: Backsubstitution I

The matrix form after the Gauss-Jordan elimination:

\[
\begin{pmatrix}
1 & a_{01} & 0 & 0 \\
0 & 1 & a_{12} & 0 \\
0 & 0 & 1 & a_{23} \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= \begin{pmatrix}
v_0 \\
v_1 \\
v_2 \\
v_3
\end{pmatrix}.
\] (59)

Solution:

\[
x_3 = v_3 \tag{60}
\]
\[
x_2 = v_2 - a_{23}x_3 \tag{61}
\]
\[
x_1 = v_1 - a_{12}x_2 \tag{62}
\]
\[
x_0 = v_0 - a_{01}x_1 \tag{63}
\]

This algorithm is known as trigonal matrix algorithm or Thomas algorithm.

- Note that the cyan colored elements do not work anything in the \(v\).
- They just become 0.
  - Just keep in mind this and never use the cyan colored elements during the back substitution to reduce computing time.
Banded Matrix

The matrix $A$ is **banded**, if it is similar to a trigonal matrix but **can have more than one nonzero elements to either side of the diagonal**, like this:

$$
A = \begin{pmatrix}
 a_{00} & a_{01} & a_{02} & 0 & 0 & 0 & 0 \\
 a_{10} & a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\
 a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\
 0 & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 \\
 0 & 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\
 0 & 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\
 0 & 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \\
\end{pmatrix}
$$

- The method to solve such equation is also similar to that for triangular matrix.
- But the backsubstitution is more complicated.
  - Such complication makes the calculation little bit slower than that for triangular matrix.
  - But still be faster than the general algorithm such as `solve` in `numpy.linalg` package.
Example 6.2: Vibration in a One-Dimensional System I

Suppose we have a set of \( N \) identical masses in a row, joined by identical linear spring as:

We ignore gravity for simplicity.
- Let \( \zeta_i \) be the displacement of the \( i \)th mass relative to its equilibrium position.
- Newton’s equation:

\[
m \frac{d^2 \zeta_i}{dt^2} = k(\zeta_{i+1} - \zeta_i) + k(\zeta_{i-1} - \zeta_i) + F_i, \tag{65}
\]

where \( m \) is the mass and \( k \) is the spring constant.
- \( F_i \) represents any external force acting on mass \( i \).
Example 6.2: Vibration in a One-Dimensional System II

- The masses at the two ends:

  \[ m \frac{d^2 \zeta_1}{dt^2} = k(\zeta_2 - \zeta_1) + F_1, \]  
  \[ m \frac{d^2 \zeta_N}{dt^2} = k(\zeta_{N-1} - \zeta_N) + F_N, \]  

  \hspace{1cm} (66) \hspace{1cm} (67)

- Assume that \( F_1 = Ce^{i\omega t} \) and \( F_i = 0 \) for all \( i > 1 \).

- By assuming that the solution \( \zeta_i = x_i e^{i\omega t} \) we obtain the \( N \)-coupled linear equations:

  \[ -m\omega^2 x_1 = k(x_2 - x_1) + C, \]  
  \[ -m\omega^2 x_i = k(x_{i+1} - x_i) + k(x_{i-1} - x_i), \]  
  \[ -m\omega^2 x_N = k(x_{N-1} - x_N), \]  

  \hspace{1cm} (68) \hspace{1cm} (69) \hspace{1cm} (70)

  where \( i \) is in the range \( 2 \leq i \leq N - 1 \).
Example 6.2: Vibration in a One-Dimensional System III

- Rearrange Eqs. (68)-(70):

\[
(\alpha - k)x_1 - kx_2 = C, \quad (71)
\]
\[
\alpha x_i - k_{i-1} - kx_{i+1} = 0, \quad (72)
\]
\[
(\alpha - k)x_N - kx_{N-1} = 0, \quad (73)
\]

where \( \alpha = 2k - m\omega^2 \).

- In matrix form:

\[
\begin{pmatrix}
(\alpha - k) & -k & & \\
- & (\alpha - k) & -k & \\
- & -k & (\alpha - k) & \\
\vdots & \ddots & \ddots & \ddots \\
- & -k & (\alpha - k) & \\
- & -k & -(\alpha - k) & \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_{N-1} \\
x_N \\
\end{pmatrix}
= \begin{pmatrix}
C \\
0 \\
0 \\
\vdots \\
0 \\
0 \\
\end{pmatrix}. \quad (74)
\]

Solve Eq. (74) with \( m = 1, k = 6, \) and \( \omega = 2 \).
Example 6-2: Solution I

Direct transform of Step1-Step5:

```python
from numpy import zeros, empty
from pylab import plot, show
from numpy.linalg import solve

# Constants
N=26
C=1.0
m=1.0
k=6.0
omega=2.0
alpha=2*k-m*omega**2

# Set up the initial values of the array
A=zeros([N,N],float)
for i in range(N-1):
    A[i,i]=alpha
    A[i,i+1]=-k
    A[i+1,i]=-k
A[0,0]=k
A[N-1,N-1]=alpha-k
v=zeros(N,float)
v[0]=C
```

Soon-Hyung Yook
Chap. 6
May 29, 2017 45 / 62
Example 6-2: Solution II

```python
# To compare the results with numpy.linalg
xx = solve(A, v)

# Perform the Gauss−Jordan Elimination
for i in range(N−1):
    # Divide row i by its diagonal element
    div = A[i, i]
    A[i, i+1] /= div
    v[i] /= div

    # Now subtract it from the next row down
    if i == N−2:
        n = 2
    else:
        n = 3
    a_tmp = A[i+1, i]
    for j in range(n):
        A[i+1, i+j] -= A[i, i+j] * a_tmp
        v[i+1] -= a_tmp * v[i]

# Divide the last element of v by the last diagonal element
v[N−1] /= A[N−1, N−1]

# Backsubstitution
```
Example 6-2: Solution III

```python
x = empty(N, float)
x[N-1] = v[N-1]
for i in range(N-2, -1, -1):
    x[i] = v[i] - A[i, i+1] * x[i+1]

# Plot the results
plot(x)
plot(x, "ko", ms=15.0)
plot(xx, "rs")
show()
```
Example 6-2: Solution IV
Example 6-2: Modified Version I

Applying the **cyan** and **red** colored parts:

```python
from numpy import zeros, empty
from pylab import plot, show
from numpy.linalg import solve

# Constants
N=26
C=1.0
m=1.0
k=6.0
omega=2.0
alpha=2*k-m*omega**2

# Set up the initial values of the array
A=zeros([N,N], float)
for i in range(N-1):
    A[i, i]=alpha
    A[i, i+1]=-k
    A[i+1, i]=-k
A[0,0]=-k
A[N-1,N-1]=alpha-k

v=zeros(N, float)
v[0]=C
```
Example 6-2: Modified Version II

```python
xx=solve(A,v)

# Perform The Gauss-Jordan Elimination
for i in range(N-1):
    # Divide row i by its diagonal element
    A[i, i+1] = A[i, i]
    v[i] = A[i, i]

    # Now subtract it from the next row down
    A[i+1, i+1] = A[i+1, i] * A[i, i+1]
    v[i+1] = A[i+1, i] * v[i]

# Divide the last element of v by the last diagonal element
v[N-1] = A[N-1, N-1]

# Backsubstitution
x=empty(N, float)
for i in range(N-1, -1, -1):
    x[i] = v[i] - A[i, i+1] * x[i+1]

# Plot the results
plot(x)
plot(x,"ko")
```
Example 6-2: Modified Version III

```python
plot(xx,"rs")
show()
```
Eigenvalues and Eigenvectors

- Eigenvalue problems are common in physics.
  - Mechanics
  - Electromagnetism
  - Quantum mechanics
  - etc.

- Most eigenvalue problems in physics concern real symmetric matrix or Hermitian matrix when complex numbers are involved.

- Focus on a real symmetric matrix $A$.

- The eigenvector $v$ satisfies:

$$Av = \lambda v,$$

(75)

where $\lambda$ is the corresponding eigenvalue.

- For $N \times N$ matrix, there are $N$ eigenvectors, $v_1, v_2, \cdots, v_N$ with eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_N$.

- Eigenvectors for symmetric matrix are orthogonal and we will assume they are normalized, i.e., $v_i \cdot v_j = \delta_{ij}$. Here $\delta_{ij}$ is Kronecker delta.
QR Decomposition

- Let $V$ be an $N \times N$ matrix whose $i$th column corresponds to the $i$th eigenvector $v_i$.
- In a matrix form Eq. (75) can be written as

$$AV = VD,$$

where $D$ is the diagonal matrix with the eigenvalues $\lambda_i$ as its diagonal entries.
- Note that the matrix $V$ is orthogonal, thus $V^T = V^{-1}$, so $V^T V = VV^T = I$.

QR Decomposition

- Like the LU decomposition, rewrite the matrix $A$ as the product $QR$, i.e.,

$$A = QR,$$

- $Q$: an orthogonal matrix
- $R$: upper-triangular matrix
Mathematics on QR Decomposition I

Suppose we have some way to calculate the matrices $Q$ and $R$.

Let $A$ be a real symmetric matrix then $A$ can be written as:

$$A = Q_1 R_1$$  \hspace{1cm} (78)

Multiplying on the left by $Q^T_1$, we get

$$Q^T_1 A = Q^T_1 Q_1 R_1 = R_1,$$  \hspace{1cm} (79)

where we use the fact that $Q_1$ is orthogonal.

Let us define a new matrix

$$A_1 = R_1 Q_1.$$  \hspace{1cm} (80)

Combining Eqs. (79) and (80), we have

$$A_1 = Q^T_1 A Q_1.$$  \hspace{1cm} (81)

Decompose $A_1$ as $A_1 = Q_2 R_2$, then $R_2 = Q^T_2 A_1$. 
Define a new matrix \( A_2 \) as

\[
A_2 = R_2 Q_2 = Q_2^T A_1 Q_2 = Q_2^T Q_1^T A Q_1 Q_2 \quad (82)
\]

Repeat the process up to total \( k \) steps then

\[
A_1 = Q_1^T A Q_1, \quad (83)
\]

\[
A_2 = Q_2^T Q_1^T A Q_1 Q_2, \quad (84)
\]

\[
A_3 = Q_3^T Q_2^T Q_1^T A Q_1 Q_2 Q_3, \quad (85)
\]

\[
\vdots \quad (86)
\]

\[
A_k = (Q_k^T \cdots Q_1^T) A (Q_1 \cdots Q_k). \quad (87)
\]

As one continue this process long enough, the matrix \( A_k \) become diagonal.

- The off-diagonal elements get smaller and smaller the more iterations of the process on do until they eventually reach zero— or as close to zero as makes no difference.
- With given accuracy we can obtain diagonalized matrix \( A_k \).
The matrix $A_k$ approximates a diagonal matrix $D$ in Eq. (76).

Let us define the additional matrix:

$$V = Q_1 Q_2 \cdots Q_k = \prod_{i=1}^{k} Q_i \tag{88}$$

Then from Eq. (87) we have

$$D = A_k = V^T AV. \tag{89}$$

Multiplying on the left by $V$:

$$AV = VD, \tag{90}$$

which is exactly the same form of Eq. (76).
Algorithm for QR Decomposition

QR Decomposition

1. Create an $N \times N$ matrix $V$ to hold the eigenvectors.
2. Initialize $V$ to be equal to the identity matrix $I$.
3. Choose a target accuracy $\epsilon$ for off-diagonal elements of the eigenvalue matrix.
4. Calculate the QR decomposition $A = QR$.
5. Update $A$ to the new value $A = RQ$.
6. Multiply $V$ on the right by $Q$.
7. Check the off-diagonal elements of $A$. If they are all less than $\epsilon$, we are done. Otherwise go back to step 4.

In numpy.linalg package, eigh() and eigvalsh() functions are also available for the general purpose.
How to Calculate $Q$ and $R$

Given $N \times N$ matrix $A$, we can compute the QR decomposition as follows:

- Let us think of the matrix as a set of $N$ column vectors $a_0, a_1, \cdots, a_{N-1}$.

\[
A = \begin{pmatrix}
| & | & | & \cdots \\
| a_0 & a_1 & a_2 & \cdots \\
| & | & | & \cdots 
\end{pmatrix}. \tag{91}
\]

- Define two new set of vectors $u_0, \cdots, u_{N-1}$ and $q_0, \cdots, q_{N-1}$ as follows (Gram-Schmidt Orthogonalization):

\[
\begin{align*}
    u_0 &= a_0, \\
    u_1 &= a_1 - (q_0 \cdot a_1)q_0, \\
    u_2 &= a_2 - (q_0 \cdot a_2)q_0 - (q_1 \cdot a_2)q_1,
\end{align*}
\]

and so forth.
How to Calculate $Q$ and $R$ II

- General form:
  \[ u_i = a_i - \sum_{j=0}^{i-1} (q_j \cdot a_i)q_j, \quad q_i = \frac{u_i}{|u_i|} \]

- Then $A$ becomes:
  \[
  A = \begin{pmatrix}
    a_0 & a_1 & a_2 & \cdots \\
    a_1 & a_2 & \cdots & \vdots \\
    a_2 & \cdots & \ddots & \vdots \\
    \vdots & \ddots & & \vdots \\
  \end{pmatrix}
  = \begin{pmatrix}
    q_0 & q_1 & q_2 & \cdots \\
    q_1 & q_2 & \cdots & \vdots \\
    q_2 & \cdots & \ddots & \vdots \\
    \vdots & \ddots & & \vdots \\
  \end{pmatrix}
  \begin{pmatrix}
    |u_0| & q_0 \cdot a_1 & q_0 \cdot a_2 & \cdots \\
    0 & |u_1| & q_1 \cdot a_2 & \cdots \\
    0 & 0 & |u_2| & \cdots \\
    0 & 0 & 0 & \vdots \\
  \end{pmatrix}
  \]

- The resulting $Q$ and $R$ have the form:
  \[
  Q = \begin{pmatrix}
    q_0 & q_1 & q_2 & \cdots \\
    q_1 & q_2 & \cdots & \vdots \\
    q_2 & \cdots & \ddots & \vdots \\
    \vdots & \ddots & & \vdots \\
  \end{pmatrix}, \quad R = \begin{pmatrix}
    |u_0| & q_0 \cdot a_1 & q_0 \cdot a_2 & \cdots \\
    0 & |u_1| & q_1 \cdot a_2 & \cdots \\
    0 & 0 & |u_2| & \cdots \\
    0 & 0 & 0 & \vdots \\
  \end{pmatrix}
  \]
Example 1

Find the eigenvalues and eigenvectors of the square matrix

\[
A = \begin{pmatrix}
1 & 4 & 8 & 4 \\
4 & 2 & 3 & 7 \\
8 & 3 & 6 & 9 \\
4 & 7 & 9 & 2
\end{pmatrix}
\]

```python
import numpy as np
from numpy.linalg import eigh

A = np.array([[1, 4, 8, 4], [4, 2, 3, 7], [8, 3, 6, 9], [4, 7, 9, 2]], float)

# Just for comparison
xx, VV = eigh(A)

print("Result using numpy.linalg")
print("xx=", xx)
print("VV=", VV)

# Implementation of QR decomposition
epsilon = 1.0e-10
n = A.shape
```
Example II

```python
N=n[1]
V=np.zeros([N,N], float)
U=np.empty([N,N], float)
Q=np.empty([N,N], float)
R=np.empty([N,N], float)

# Initialize V
for i in range(N):
    V[i,i]=1.0

delta=1.0
while delta>epsilon:
    for i in range(N):
        U[:,i]=A[:,i]
        if i>0:
            for j in range(i):
                U[:,i]−=(np.dot(Q[:,j], A[:,i])*Q[:,j])
            magU=np.dot(U[:,i],U[:,i])**((1/2))
            Q[:,i]=U[:,i]/magU

# Computing R matrix
for j in range(N):
    for k in range(N):
        if j>k:
            R[j,k]=0
```
Example III

```python
elif j==k:
    R[j, k] = np.dot(U[:, j], U[:, j])** (1/2)
else:
    R[j, k] = np.dot(Q[:, j], A[:, k])

# print("R=", R)
A= np.dot(R, Q)
V= np.dot(V, Q)
delta = 0.0
for j in range(N):
    for k in range(N):
        if j<k:
            if delta < abs(A[j, k]):
                delta = abs(A[j, k])
            # print("delta=", delta)
            # input()
x= np.empty(N, float)
for i in range(N):
    x[i] = A[i, i]

print("\n--- Result obtained from my QR decomposition code ---")
print("x=", x)
print(V)
```