1. Upload the notebook of week three.

03.26

solve system of linear equation

problem 1

```
# Ax = B

#  T1 + m1 * a = m1 * g * (np.sin(theta) - mu1 * np.cos(theta))
# -T1 + T2 + m2 * a = m2 * g * (np.sin(theta) - mu2 * np.cos(theta))
# -T2 + T3 + m3 * a = m3 * g * (np.sin(theta) - mu3 * np.cos(theta))
# -T3 + m4 * a = - m4 * g

# m = [10, 4, 5, 6]
# mu = [0.25, 0.3, 0.2]
# g = 9.8 m/s^2

# Find A, B
```

In [1]:

```python
import numpy as np

mass = np.array([10, 4, 5, 6]) # kg
mu = np.array([0.25, 0.3, 0.2])
gval = 9.8 # m / s^2
theta = np.pi/4

aa = np.array([[ 1, 0, 0, mass[0]],
               [-1, 1, 0, mass[1]],
               [ 0,-1, 1, mass[2]],
               [ 0, 0,-1, mass[3]]])

def f(x,i):
    return (np.sin(x)-mu[i]*np.cos(x))

bb = np.array([ mass[0]*gval*f(theta,0),
               mass[1]*gval*f(theta,1),
               mass[2]*gval*f(theta,2),
              -mass[3]*gval])

np.linalg.solve(aa,bb)
```

Out [1]: array([35.85477069, 48.81074968, 68.47054664,  1.61175777])

T1 = 35.85477069, T2 = 48.81074968, T3 = 68.47054664, a = 1.61175777 problem 2

```
# x1 * (k1 + k2) - (x2 - x1) * k3 - (x3 - x1) * k5 - W1 = 0
# k3 * (x2 - x1) - k4 * (x3 - x2) - W2 = 0
# k4 * (x3 - x2) + k5 * (x3 - x1) - W3 = 0
```

In [2]:

```
k = [1,2,3,4,5]
W = [1,2,3]

aa = np.array([[k[0]+k[1]+k[2]+k[4],    -k[2],     -k[4]],
               [                -k[2],k[2]+k[3],     -k[3]],
               [                -k[4],    -k[3],k[3]+k[4]]])

bb = np.array(W)

np.linalg.solve(aa,bb)
```

Out [2]: array([2.        , 2.63829787, 2.61702128])

x1 = 2, x2 = 2.63829787, x3 = 2.61702128

03.27

solving linear equation algorithm

In [1]:

```
import numpy as np

a = np.array([[ 4.,-2., 1.],
              [-2., 4.,-2.],
              [ 1.,-2., 4.]])
b = np.array([11.,-16.,17.])
x = np.linalg.solve(a,b)
ainv = np.linalg.inv(a)
x
```

Out [1]: array([ 1., -2., 3.])

In [2]:

```
np.matmul(a,x) # Ax = B
```

Out [3]: array([ 11., -16., 17.])

In [4]:

```
np.matmul(ainv,b) # A^(-1)Ax = B
```

Out [4]: array([ 1., -2., 3.])

Gauss elimination
In [5]:

```
# Initializing parameters
ipiv    = 0
nsize   = 3
bb      = np.zeros(nsize)

d = np.array(a)
f = np.array(b)

# eliminate 2nd row
rnum    = 1
lam     = a[ipiv+rnum,ipiv]/a[ipiv,ipiv]
c       = a[ipiv+rnum,0:nsize] - lam * a[ipiv,0:nsize]
d[rnum] = c
f[rnum] = f[rnum] - lam * f[ipiv]
print(d,'\n',f,'\n')

# eliminate 3rd row
rnum    = 2
lam     = a[ipiv+rnum,ipiv]/a[ipiv,ipiv]
c       = a[ipiv+rnum,0:nsize] - lam * a[ipiv,0:nsize]
d[rnum] = c
f[rnum] = f[rnum] - lam * f[ipiv]
print(d,'\n',f,'\n')

# eliminate 2nd column 3nd row
ipiv    = 1

rnum    = 2
lam     = d[rnum,ipiv]/d[ipiv,ipiv]
c       = d[rnum,0:nsize] - lam * d[ipiv,0:nsize]
d[rnum] = c
f[rnum] = f[rnum] - lam * f[ipiv]
print(d,'\n',f)
```

Out [5]:

```
[[ 4.  -2.   1. ]
 [ 0.   3.  -1.5]
 [ 1.  -2.   4. ]]
 [ 11.  -10.5  17. ]

[[ 4.   -2.    1.  ]
 [ 0.    3.   -1.5 ]
 [ 0.   -1.5   3.75]]
 [ 11.   -10.5   14.25]

[[ 4.  -2.   1. ]
 [ 0.   3.  -1.5]
 [ 0.   0.   3. ]]
 [ 11.  -10.5   9. ]
```

Cholesky decomposition

Ax = b  =>  LUx = b  (A = LU, transpose(L) = U)

Ly = b  =>  Ux = y

In [6]:

```
chole  = np.linalg.cholesky(a)
cholet = np.transpose(chole)
amat   = np.matmul(chole,cholet)

print(chole,'\n\n',cholet,'\n\n',amat)
```

Out [6]:

```
[[ 2.          0.          0.        ]
 [-1.          1.73205081  0.        ]
 [ 0.5        -0.8660254   1.73205081]]

 [[ 2.         -1.          0.5       ]
 [ 0.          1.73205081 -0.8660254]
 [ 0.          0.          1.73205081]]

 [[ 4. -2.  1.]
 [-2.  4. -2.]
 [ 1. -2.  4.]]
```

# Use the Cholesky matrix to obtain y, and then x

In [7]:

```
chole  = np.linalg.cholesky(a)
cholet = np.transpose(chole)

nsize = 3

# L*y = b
ys = np.zeros(nsize)
ys[0] = b[0]/chole[0,0]
ys[1] = ( b[1] - chole[1,0]*ys[0] ) / chole[1,1]
ys[2] = ( b[2] - chole[2,0]*ys[0] - chole[2,1]*ys[1]) / chole[2,2]
print(ys)

# U*x = y
xs    = np.zeros(nsize)
xs[2] = ys[2]/cholet[2,2]
xs[1] = ( ys[1] - cholet[1,2]*xs[2] ) / chole[1,1]
xs[0] = ( ys[0] - cholet[0,1]*xs[1] - cholet[0,2]*xs[2]) / cholet[0,0]
print(xs)
```

Out [7]:

```
[ 5.5        -6.06217783  5.19615242]
[ 1. -2.  3.]
```

2. Write a code for the gauss elimination method and apply it to solve a non-singular matrix. Verify the correctness of the result comparing with linalg.solve

HW3_2.py

```
import numpy as np

def triangular(A,B):
    # Initializing parameters
    nsize   = len(A)
    a = np.array(A)
    b = np.array(B)

    for ipiv in range(0,nsize-1):
        for rnum in range(ipiv+1,nsize):
            lam    = a[rnum,ipiv]/a[ipiv,ipiv]
            a[rnum] = a[rnum,0:nsize] - lam * a[ipiv,0:nsize]
            b[rnum] = b[rnum] - lam * b[ipiv]

    return [a,b]

def solve(A,B):
    # Initializing parameters
    nsize = len(A)
    a    = np.array(A)
    b    = np.array(B)

    # Triangular
    for ipiv in range(0,nsize-1):
        for rnum in range(ipiv+1,nsize):
            lam    = a[rnum,ipiv]/a[ipiv,ipiv]
            a[rnum] = a[rnum,0:nsize] - lam * a[ipiv,0:nsize]
            b[rnum] = b[rnum] - lam * b[ipiv]

    x = np.zeros(nsize)
    x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
    for n in range(2,nsize+1):
        sum, i = 0, nsize - n
        for j in range(i,nsize):
            sum = sum + a[i][j] * x[j]
        x[i] = (b[i] - sum) / a[i][i]

    return x

def solve0(A,B): # Use triangular function
    # Initializing parameters
    nsize = len(A)
    C = triangular(A,B)
    a = C[0]
    b = C[1]

    x = np.zeros(nsize)
    x[nsize-1]=b[nsize-1]/a[nsize-1][nsize-1]
    for n in range(2,nsize+1):
        sum, i = 0, nsize - n
        for j in range(i,nsize):
            sum = sum + a[i][j] * x[j]
        x[i] = (b[i] - sum) / a[i][i]

    return x
```

```
$ python
>>> from HW3_2 import *
>>> import numpy as np
>>> a = np.array([[4,-2,1],[-2,4,-2],[1,-2,4]])
>>> b = np.array([11,-16,17])
>>> solve(a,b)
>>> array([ 1., -2.,  3.])
>>> np.linalg.solve(a,b)
>>> array([ 1., -2.,  3.])
```

error is too small because a and b is integer

testing random 4*4 matrix

test.py

```python
import numpy as np
from HW3_2 import *

a, b = [], []

A = np.array([[0.,0.,0.,0.],[0.,0.,0.,0.],[0.,0.,0.,0.],[0.,0.,0.,0.]])

# Make 10 random array b
for i in range(0,10):
    b.append(np.random.random(4))

for i in range(0,10):
    a_sample = np.random.random(16)

    for j in range(0,4):
        A[j,0:4] = a_sample[j*4:(j+1)*4]

    a.append(A)

num = 0
results = np.zeros(4)
for i in range(0,10):
    x = np.linalg.solve(a[i],b[i])
    solve = solve0(a[i],b[i])
    results += x - solve
    num =+ 1

print(results/num)
```

```
$ python test.py
[-2.91433544e-15 -1.72084569e-15  1.04916076e-14 -5.05151476e-15]
```

error is very small.

3. Write the backward and forward substitution codes to obtain y and then x, for the solutions of Ax=b, where A=LU, such that Ux=y, and Ly=LUx=b. Use linalg.cholesky to generate the L and U matrices, and verify against the solutions in problem 2.

In [8]:

```
def chole(a,b):
    L = np.linalg.cholesky(a)
    U = np.transpose(L)
    nsize = len(a)

    y = np.linalg.solve(L,b)
    x = np.linalg.solve(U,y)

    return x

print(solve(a,b) - np.linalg.solve(a,b))
print(chole(a,b) - np.linalg.solve(a,b))
```

Out [8]:
[0. 0. 0.]
[0.0000000e+00 0.0000000e+00 4.4408921e-16]

Cholesky algorithm’s error is higher than gauss elimination
error ware maked np.linalg.cholesky() function.

I coudn’t make random 4*4 positive definite matrix, so coudn’t get general error