# 66310837

## นายจิรัฐ ฟองดา

```
import numpy as np
import numpy.linalg as la
import scipy.linalg as sla

A = np.random.rand(2,2)
print(A)
la.cond(A)
```

```
[[0.2796854  0.07291289]
 [0.33839441 0.02180566]]
```

```
np.float64(10.593670482229864)
```

```
la.cond([
         [0.000001, 0.1],
         [0,   1]
         ], 2)
```

```
np.float64(1010000.0000000097)
```

```
ndim = np.array([2,3,8,11,14])

for nd in ndim:
    ## This is the vector 'x' that we want to obtain (the exact one)
    x = np.ones(nd)
    ## Creat a matrix with random values between 0 and 1
    A = np.random.rand(nd,nd)
    ## We compute the matrix-vector multiplication
    ## to find the right-hand side b
    b = A @ x
    ## We now use the linear algebra pack to compute Ax = b and solve
for x
    X_solve = la.solve(A,b)
    ## What do we expect?
    print("------ N =", nd, "---------")
    error = X_solve-x
    print("Norm of error = ", la.norm(error,2))
```

```
------ N = 2 ---------
Norm of error =  0.0
------ N = 3 ---------
Norm of error =  1.1102230246251565e-16
------ N = 8 ---------
Norm of error =  4.2826412465193164e-15
```

```
------ N = 11 ---------
Norm of error =  9.774377279052153e-15
------ N = 14 ---------
Norm of error =  5.23184869964764e-14
```

```
X_solve
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
def Hilbert(n):

    H = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            H[i,j] = 1.0/(j+i+1)
    return H
```

```
for nd in ndim:
    ## This is the vector 'x' that we want to obtain (the exact one)
    x = np.ones(nd)
    ## Creat the Hilbert matrix
    A = Hilbert(nd)
    ## We compute the matrix-vector multiplication
    ## to find the right-hand side b
    b = A @ x
    ## We now use the linear algebra pack to compute Ax = b and solve
for x
    X_solve = la.solve(A,b)
    ## What do we expect?
    print("------ N =", nd, "---------")
    error = X_solve-x
    print("Norm of error = ", la.norm(error,2))
```

```
------ N = 2 ---------
Norm of error =  8.005932084973442e-16
------ N = 3 ---------
Norm of error =  1.4464463460722808e-14
------ N = 8 ---------
Norm of error =  1.9073830647724366e-07
------ N = 11 ---------
Norm of error =  0.003753868806774399
------ N = 14 ---------
Norm of error =  8.248637159750212
```

```
X_solve
```

```
array([ 0.99999986,  1.00001865,  0.99937137,  1.00902718,
0.9321984 ,
        1.28905897,  0.31121478,  1.67829176,  1.8448608 , -
2.71520824,
        6.48491627, -3.34820964,  2.84412601,  0.67033372])
```

```
for nd in ndim:
    ## This is the vector 'x' that we want to obtain (the exact one)
    x = np.ones(nd)
    ## Creat the Hilbert matrix
    A = Hilbert(nd)
    ## We compute the matrix-vector multiplication
    ## to find the right-hand side b
    b = A @ x
    ## We now use the linear algebra pack to compute Ax = b and solve
for x
    X_solve = la.solve(A,b)
    ## What do we expect?
    print("------ N =", nd, "---------")
    error = X_solve-x
    print("Norm of error = ", la.norm(error,2))
    print("Condition number = ", la.cond(A,2))
```

```
------ N = 2 ---------
Norm of error =  8.005932084973442e-16
Condition number =  19.281470067903967
------ N = 3 ---------
Norm of error =  1.4464463460722808e-14
Condition number =  524.0567775860627
------ N = 8 ---------
Norm of error =  1.9073830647724366e-07
Condition number =  15257575568.347378
------ N = 11 ---------
Norm of error =  0.003753868806774399
Condition number =  522477970426706.44
------ N = 14 ---------
Norm of error =  8.248637159750212
Condition number =  3.213787720967528e+17
```

```
for nd in ndim:
    ## This is the vector 'x' that we want to obtain (the exact one)
    x = np.ones(nd)
    ## Creat the Hilbert matrix
    A = Hilbert(nd)
    ## We compute the matrix-vector multiplication
    ## to find the right-hand side b
    b = A @ x
    ## We now use the linear algebra pack to compute Ax = b and solve
for x
    X_solve = la.solve(A,b)
    ## What do we expect?
    print("------ N =", nd, "---------")
    error = X_solve - x
    residual = A@X_solve - b
    print("Error norm = ", la.norm(error,2))
```

```
    print("Residual norm = ",la.norm(residual,2))
    print("Condition number = ", la.cond(A,2))
```

```
------ N = 2 ---------
Error norm =  8.005932084973442e-16
Residual norm =  0.0
Condition number =  19.281470067903967
------ N = 3 ---------
Error norm =  1.4464463460722808e-14
Residual norm =  0.0
Condition number =  524.0567775860627
------ N = 8 ---------
Error norm =  1.9073830647724366e-07
Residual norm =  6.568167990716596e-16
Condition number =  15257575568.347378
------ N = 11 ---------
Error norm =  0.003753868806774399
Residual norm =  3.3306690738754696e-16
Condition number =  522477970426706.44
------ N = 14 ---------
Error norm =  8.248637159750212
Residual norm =  6.933340566559918e-16
Condition number =  3.213787720967528e+17
```

```
for nd in ndim:
    ## This is the vector 'x' that we want to obtain (the exact one)
    x = np.ones(nd)
    ## Creat the Hilbert matrix
    A = Hilbert(nd)
    ## We compute the matrix-vector multiplication
    ## to find the right-hand side b
    b = A @ x
    ## We now use the linear algebra pack to compute Ax = b and solve
for x
    X_solve = la.solve(A,b)
    ## What do we expect?
    print("------ N =", nd, "---------")
    error = X_solve-x
    residual = A@X_solve - b
    print("Error norm = ", la.norm(error,2))
    # print("Residual norm = ",la.norm(residual,2))
    print("|dx|/|x| < ", la.cond(A,2)*10**(-16))
    # print("Condition number = ", la.cond(A,2))
```

```
------ N = 2 ---------
Error norm =  8.005932084973442e-16
|dx|/|x| <  1.928147006790397e-15
------ N = 3 ---------
Error norm =  1.4464463460722808e-14
|dx|/|x| <  5.240567775860627e-14
```

```
------ N = 8 ---------
Error norm =  1.9073830647724366e-07
|dx|/|x| <  1.5257575568347377e-06
------ N = 11 ---------
Error norm =  0.003753868806774399
|dx|/|x| <  0.05224779704267064
------ N = 14 ---------
Error norm =  8.248637159750212
|dx|/|x| <  32.137877209675274
```

```
def myLU(A):
    n = A.shape[0]
    U = np.zeros((n,n))
    L = np.zeros((n,n))
    M = A.copy()
    for i in range(n):
        U[i,i:] = M[i,i:]
        L[i:,i] = M[i:,i]/U[i,i]
        M[i+1:,i+1:] -= np.outer(L[i+1:,i],U[i,i+1:])
    return L,U

# Creating the arrays
c = 1e-16
A = np.array([[c, 1.], [-1, 1]])
# xx is the exact solution
xx = np.array([1,1])
b = A.dot(xx)

# Comput the LU
L,U = myLU(A)

# Solve
# x is the numerical (xhat)
y = sla.solve_triangular(L, b, lower=True)
x = sla.solve_triangular(U, y)

x = sla.solve(A,b)

print("Exact solution = ", xx)
print("Computed solution = ",x)
print("Error = ", la.norm(xx-x))
print("Condition number = ", la.cond(A,2))
print("Resiudal = ", la.norm(A@x - b))
```

```
Exact solution =  [1 1]
Computed solution =  [1. 1.]
Error =  0.0
Condition number =  2.6180339887498953
Resiudal =  0.0
```