Lab 03. Solve linear system and matrix decompositions

Moahmed Stifi

2IA

1 Code

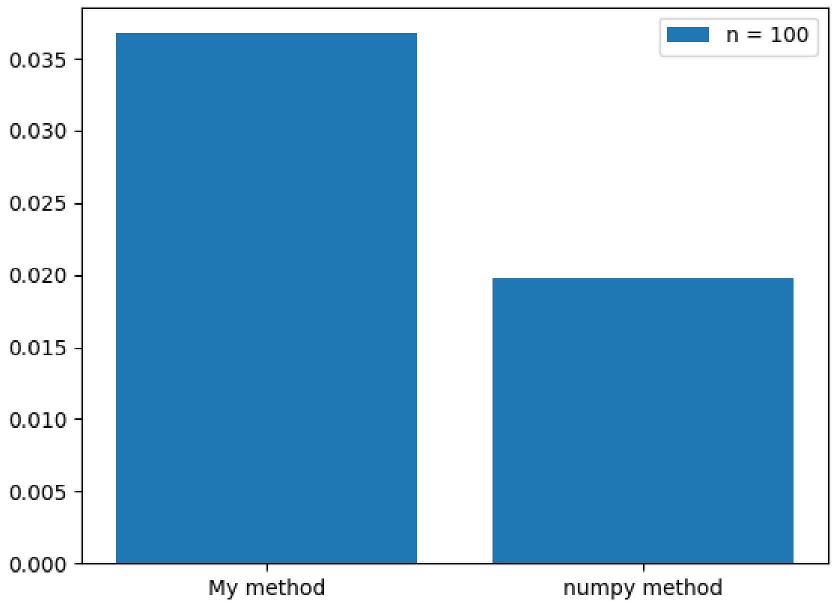
1.1 The elimination of Gauss-Jordan to Solve a system S of linear

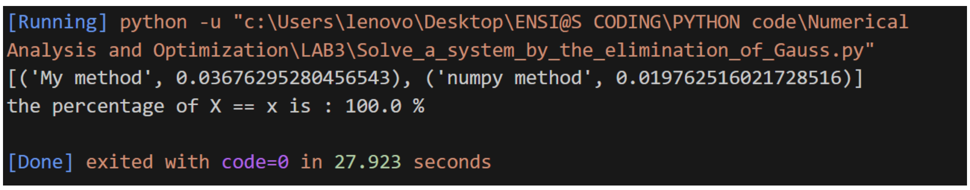
Use the elimination of Gauss-Jordan to Solve a system S of linear equations A x = b defined by the matrix A and the vector b.
code file : `Solve_a_system_by_the_elimination_of_Gauss.py`

```python
import numpy as np
from time import time
import matplotlib.pyplot as plt
np.random.seed(42)
def for_Elimination(A,b):
    n = A.shape[0]
    arr = np.zeros((n, n+1))
    arr[:,:n] = A.copy()
    arr[:,n] = b.copy()
    for i in range(n-1):
        p_ind = i + np.argmax(np.abs(arr[i:,i]))
        arr[i,i:], arr[p_ind,i:] = arr[p_ind,i:].copy(),arr[i,i:].copy
            ↪ ()
        pivot = arr[i,i]
        if pivot == 0 : print("le systeme n admet pas de solution
            ↪ unique ")
        for k in range(i+1, n):
            m = arr[k,i]/pivot
            arr[k,i:] -= m*arr[i, i:]
    return arr[:,:n], arr[:,n]

def back_Substitution(A,b):
    n = A.shape[0]
    X = np.empty(n)
    X[-1] = b[-1]/A[-1,-1]
    for i in range(-2, -(n+1), -1):
        X[i] = (b[i] - np.dot(A[i,i+1:], X[i+1:]))/A[i,i]

    return X

def Solve(A,b):
    """
     Solve a system S of linear equations A x = b defined \
        by the matrix A and the vector b
    """
    a, bb = for_Elimination(A,b)
```

```python
    return back_Substitution(a, bb)
if __name__ == "__main__":
    n = 100
    A = np.random.randn(n,n)

    b = np.random.randn(n)

    names = ["My method", "numpy method"]
    t = []
    # ------------------------------------------------------------
    t1 =time(); X = Solve(A,b);t.append(time()-t1)
    t1 =time(); x = np.linalg.solve(A,b);t.append(time()-t1)
    # ------------------------------------------------------------
    plt.bar(names,t, label = "n = "+str(n))
    plt.legend()
    plt.show()
    # ------------------------------------------------------------
    print(list(zip(names,t)))
    print(f"the percentage of X == x is : {np.sum(np.around(X,9) == np.around(x,9))*100/n} % ")
```

1.1.1 Screenshots





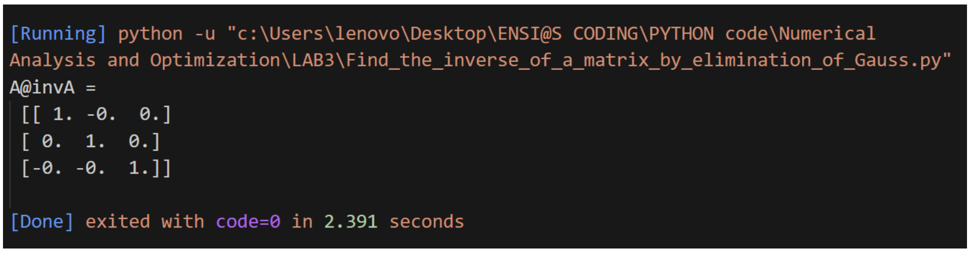
```
[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical Analysis and Optimization\LAB3\Solve_a_system_by_the_elimination_of_Gauss.py"
[('My method', 0.03676295280456543), ('numpy method', 0.019762516021728516)]
the percentage of X == x is : 100.0 %

[Done] exited with code=0 in 27.923 seconds
```

1.2 Use the elimination of Gauss-Jordan to find the inverse of a matrix A

code file : `Find_the_inverse_of_a_matrix_by_elimination_of_Gauss.py`

```python
from Solve_a_system_by_the_elimination_of_Gauss import Solve
import numpy as np
def Inverse(A):
    n = A.shape[0]
    inv_A = np.empty((n,n))
    b = np.array([0]*n)
    for i in range(n):
        b[i] = 1
        inv_A[:,i] = Solve(A,b)
        b[i] = 0

    return inv_A
if __name__ == "__main__" :
    A = np.array([[25,5,1],
                  [64,8,1],
                  [144,12,1]])

    invA = Inverse(A)
    print("A@invA = \n",np.around(A@invA))
```

1.2.1 Screenshots

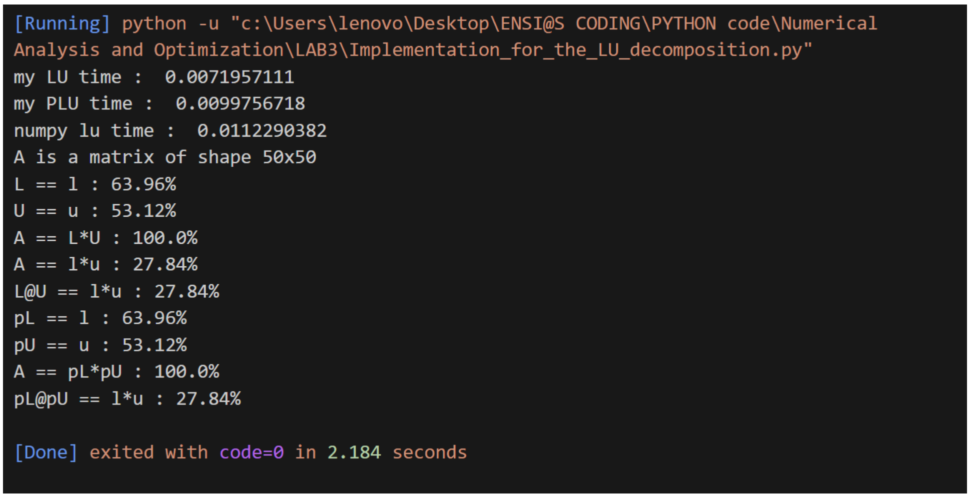


1.3 Python implementation for the LU decomposition.

code file : `Implementation_for_the_LU_decomposition.py`

```python
import numpy as np
import time as t
import scipy.linalg
np.random.seed(42)
# Define the matrix
n = 50
A = np.random.randn(n,n)
# ----------| implementation for the LU decomposition|-------
def LU(A, ameliorer  = False) :
    n = len(A)
    U = A.copy()
    L = np.eye(n)
    for i in range(n-1):

        pivot =  U[i,i]
        if pivot == 0 : print("pivot = 0") ; return L,U

        for k in range(i+1, n):
            L[k,i] = U[k,i]/pivot
```

```python
                U[k] = U[k] - L[k,i]*U[i]
    return L,U

def PLU(A):
    n = A.shape[0]
    U = A.copy()
    L = np.eye(n, dtype= np.double)
    P = np.eye(n, dtype= np.double)

    for i in range(n):
        for k in range(i,n):
            if ~np.isclose(U[i,i], 0) :
                break
            U[[k,k+1]] = U[[k+1, k]]
            P[[k,k+1]] = P[[k+1, k]]

        for k in range(i+1, n):
            L[k,i] = U[k,i]/U[i,i]
            U[k] = U[k] - L[k,i]*U[i]
    return P,L,U



if __name__ == "__main__":
    # ------------------------------------------------------------
    t1 = t.time() ; L,U = LU(A) ;t1 = t.time() - t1
    t2 = t.time();p, l,u = scipy.linalg.lu(A);t2 = t.time() - t2
    t3 = t.time();p, pL, pU = PLU(A) ; t3 = t.time() - t3
    # ------------------------------------------------------------
    print("my LU time : ",np.around(t1,10))
    print("my PLU time : ",np.around(t3,10))
    # ------------------------------------------------------------
    print("numpy lu time : ",np.around(t2,10))
    # ------------------------------------------------------------
    print(f"A is a matrix of shape {n}x{n}")
    print(f"L == l : {np.sum(np.around(l) == np.around(L))*100/len(l)**2}%")
    print(f"U == u : {np.sum(np.around(u) == np.around(U))*100/len(l)**2}%")
    print(f"A == L*U : {np.sum(np.around(A) == np.around(L@U))*100/len(l)**2}%")
    print(f"A == l*u : {np.sum(np.around(A) == np.around(l@u))*100/len(l)**2}%")
    print(f"L@U == l*u : {np.sum(np.around(L@U) == np.around(l@u))*100/len(l)**2}%")

    print(f"pL == l : {np.sum(np.around(l) == np.around(pL))*100/len(l)**2}%")
    print(f"pU == u : {np.sum(np.around(u) == np.around(pU))*100/len(l)**2}%")
    print(f"A == pL*pU : {np.sum(np.around(A) == np.around(pL@pU))*100/len(l)**2}%")
    print(f"pL@pU == l*u : {np.sum(np.around(pL@pU) == np.around(l@u))*100/len(l)**2}%")
```



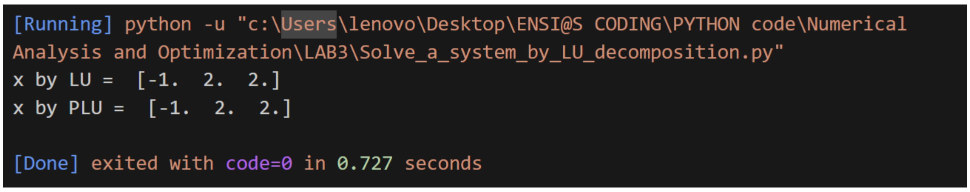
1.3.1 Screenshots

1.4 Use the LU decompositions to solve the system A x = b

code file : `Solve_a_system_by_LU_decomposition.py`

```python
from Implementation_for_the_LU_decomposition import LU, PLU
import numpy as np

def Solve_Lz_b(L,b):
    n = L.shape[0]
    Z = np.empty(n)
    Z[0] = b[0]/L[0,0]
    for i in range(1,n):
        Z[i] = (b[i] - np.dot(L[i,:i], Z[:i]))/L[i,i]
    return Z

def Solve_Ux_z(U,z):
    n = U.shape[0]
    X = np.empty(n)
    X[-1] = z[-1]/U[-1,-1]
    for i in range(-2, -(n+1), -1):
        X[i] = (z[i] - np.dot(U[i,i+1:], X[i+1:]) )/U[i,i]
    return X

def Solve(L,U ,b):
    z = Solve_Lz_b(L,b)
    x = Solve_Ux_z(U,z)
    return x
if __name__ == "__main__":
    A = np.array([[2,4,-2],
                  [4,9,-3],
                  [-2,-3,7]])
    b = np.array([2,8,10])

    L,U = LU(A)
    Xlu = Solve(L,U,b)
    print("x by LU = ",Xlu)

    P,L,U = PLU(A)
    Xlu = Solve(L,U,np.dot(P,b))
    print("x by PLU = ",Xlu)
```

1.4.1 Screenshots



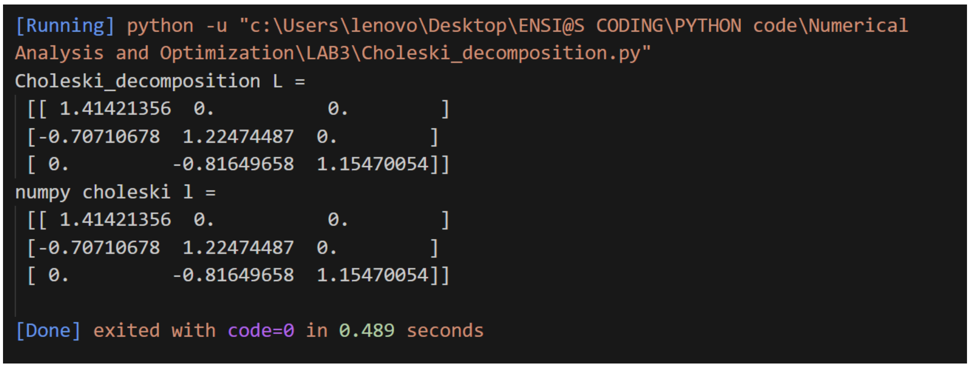
```
[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\Solve_a_system_by_LU_decomposition.py"
x by LU =  [-1.  2.  2.]
x by PLU =  [-1.  2.  2.]

[Done] exited with code=0 in 0.727 seconds
```

1.5 Python implementation for the Choleski's decomposition

code file : `Choleski_decomposition.py`

```python
import numpy as np

def Choleski_decomposition(A):
    L = np.zeros(A.shape, dtype= np.double)
    n = A.shape[0]
    for k in range(n):
        L[k,k] = np.sqrt(A[k,k] - np.dot(L[k,:k], L[k,:k]))

        for i in range(k+1, n):
            L[i,k] = ( A[i,k] - np.dot(L[i,:k], L[:k,k]) )/L[k,k]

    return L
if __name__ == "__main__":

    A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)


    L = Choleski_decomposition(A)
    l = np.linalg.cholesky(A)
    print("Choleski_decomposition L = \n",L)
    print("numpy choleski l = \n",l)
```

1.5.1 Screenshots



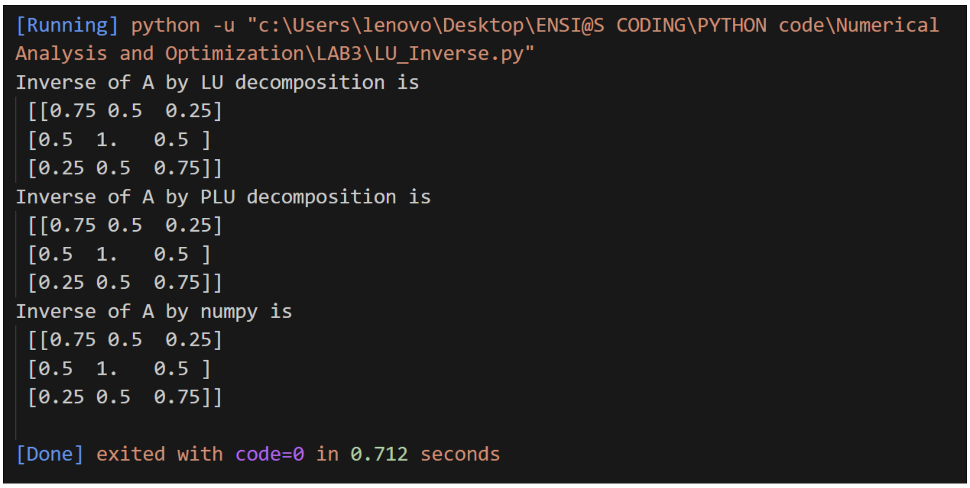
```
[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\Choleski_decomposition.py"
Choleski_decomposition L =
 [[ 1.41421356  0.          0.        ]
 [-0.70710678  1.22474487  0.        ]
 [ 0.         -0.81649658  1.15470054]]
numpy choleski l =
 [[ 1.41421356  0.          0.        ]
 [-0.70710678  1.22474487  0.        ]
 [ 0.         -0.81649658  1.15470054]]

[Done] exited with code=0 in 0.489 seconds
```

1.6 Use the LU decomposition to find the inverse matrix of A

```python
from Solve_a_system_by_LU_decomposition import Solve_Lz_b as for_sub
from Solve_a_system_by_LU_decomposition import Solve_Ux_z as back_sub
from Implementation_for_the_LU_decomposition import LU, PLU

```

```python
import numpy as np

def LU_Inverse(L,U, P = None):
    n = L.shape[0]
    Inv = np.empty((n,n), dtype=np.double)
    x = np.empty(n, dtype=np.double)
    b = np.zeros(n, dtype=np.double)
    for i in range(n):
        b[i] = 1
        # Solve the equation L * X = b for X using forward substitution
        if P is not None:
            x = for_sub(L,np.dot(P,b))
        else :
            x = for_sub(L, b)
        # Solve the equation L^T * Y = X for Y using back substitution
        x = back_sub(U, x)
        # Set the i-th column of the inverse matrix
        Inv[:,i] = x
        b[i] = 0
    return Inv

if __name__ == "__main__":
    A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)

    L, U = LU(A)

    LU_InvA = LU_Inverse(L,U)
    P,L,U = PLU(A)
    PLU_InvA = LU_Inverse(L,U, P)
    numpy_invA = np.linalg.inv(A)
    print(f"Inverse of A by LU decomposition is \n {LU_InvA}")
    print(f"Inverse of A by PLU decomposition is \n {PLU_InvA}")
    print(f"Inverse of A by numpy is \n {numpy_invA}")
```

1.6.1 Screenshots

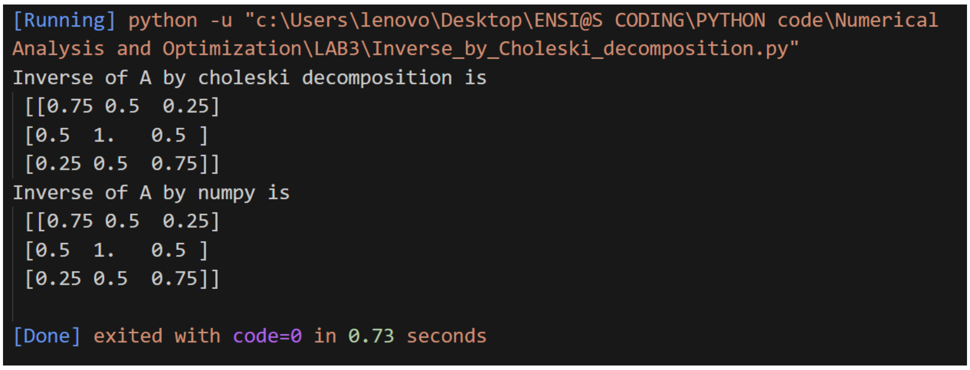


1.7 Use the Choleski's decomposition to find the inverse matrix of A

code file : `Inverse_by_Choleski_decomposition.py`

```python
from Choleski_decomposition import Choleski_decomposition
from Solve_a_system_by_LU_decomposition import Solve_Lz_b as for_sub
from Solve_a_system_by_LU_decomposition import Solve_Ux_z as back_sub
import numpy as np

def Inverse_by_choleski(L):
    n = L.shape[0]
    Inv = np.empty((n,n), dtype=np.double)
    x = np.empty(n, dtype=np.double)
    b = np.zeros(n, dtype=np.double)
    for i in range(n):
        b[i] = 1
        # Solve the equation L * X = b for X using forward substitution
        x = for_sub(L, b)
        # Solve the equation L^T * Y = X for Y using back substitution
        x = back_sub(L.T, x)
        # Set the i-th column of the inverse matrix
        Inv[:,i] = x
        b[i] = 0
    return Inv

    return Inv
if __name__ == "__main__":
    A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)

    L = Choleski_decomposition(A)
    Choleski_InvA = Inverse_by_choleski(L)
    numpy_invA = np.linalg.inv(A)
    print(f"Inverse of A by choleski decomposition is \n {Choleski_InvA}")
    print(f"Inverse of A by numpy is \n {numpy_invA}")
```

1.7.1 Screenshots



1.8 Comparing these solving methods

code file : `comparing_these_solving_methods.py`

```python
from Find_the_inverse_of_a_matrix_by_elimination_of_Gauss import Inverse as Gauss_Inv
from Choleski_decomposition import Choleski_decomposition
from Inverse_by_Choleski_decomposition import Inverse_by_choleski as Choleski_Inv
```

```python
from Solve_a_system_by_LU_decomposition import Solve as LU_solve
from Solve_a_system_by_the_elimination_of_Gauss import Solve as
    ↪ Gauss_solve
from Implementation_for_the_LU_decomposition import LU, PLU
from LU_Inverse import LU_Inverse as LU_Inv
from time import time
import matplotlib.pyplot as plt
import numpy as np
np.random.seed = 42
n = 10
names = ["LU_inv", "PLU_inv", "numpy_inv", "Gauss_inv",
         "Choleski_inv", "LU_solve",
         "PLU_solve", "Gauss_solve", "numpy_solve"]
times = []

# A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)
A = np.random.randn(n,n)
b = np.random.randn(n)
# ----------------------------------------------------
t = time()
l,u = LU(A)
LU_Inv(l,u)
times.append(time()- t)
# ----------------------------------------------------
t = time()
p,l,u = PLU(A)
LU_Inv(l,u,p)
times.append(time()- t)
# ----------------------------------------------------
t = time()
np.linalg.inv(A)
times.append(time()- t)
# ----------------------------------------------------
t = time()
Gauss_Inv(A)
times.append(time()- t)
# ----------------------------------------------------
t = time()
l = Choleski_decomposition(A)
Choleski_Inv(l)
times.append(time()- t)
# ----------------------------------------------------
t = time()
l,u = LU(A)
LU_solve(l,u,b)
times.append(time()- t)
# ----------------------------------------------------
t = time()
p,l,u = PLU(A)
LU_solve(l,u, np.dot(p,b))
times.append(time()- t)
# ----------------------------------------------------
t = time()
Gauss_solve(A,b)
times.append(time()- t)
# ----------------------------------------------------
t = time()
np.linalg.solve(A,b)
```

```
times.append(time()- t)
# ---------------------------------------------------

plt.bar(names[:-4], times[:-4], alpha = 0.6 )
plt.bar(names[-4:], times[-4:], alpha = 0.6 )
plt.title("runing time of these method")
plt.xlabel("methods names")
plt.ylabel("time (s)")
plt.show()
```

1.8.1 Screenshots