

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 $Ax = b$ defined by the matrix A and the vector b.

code file : Solve_a_system_by_the_elimination_of_Gauss.py

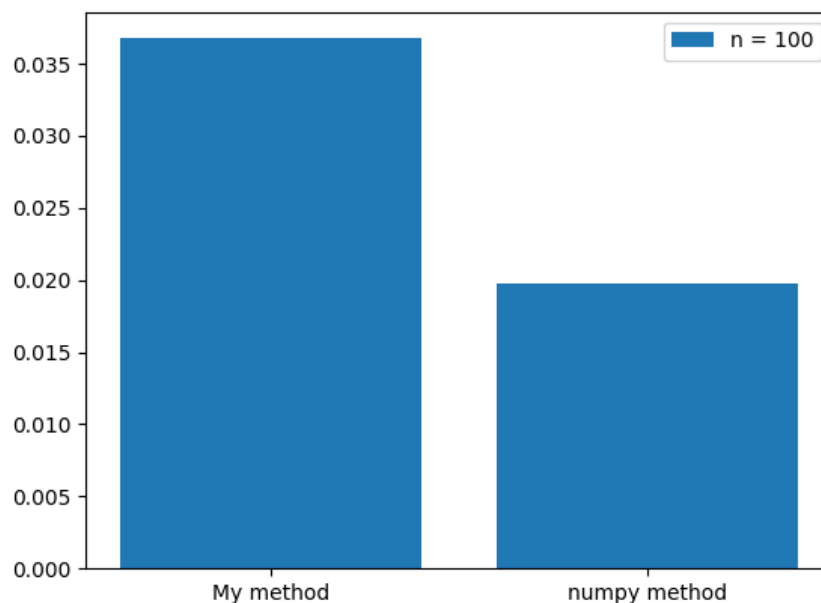
```
1 import numpy as np
2 from time import time
3 import matplotlib.pyplot as plt
4 np.random.seed(42)
5 def for_Elimination(A,b):
6     n = A.shape[0]
7     arr = np.zeros((n, n+1))
8     arr[:, :n] = A.copy()
9     arr[:, n] = b.copy()
10    for i in range(n-1):
11        p_ind = i + np.argmax(np.abs(arr[i:, i]))
12        arr[i, i:], arr[p_ind, i:] = arr[p_ind, i:].copy(), arr[i, i:].copy()
13        pivot = arr[i, i]
14        if pivot == 0 : print("le systeme n admet pas de solution \n unique ")
15        for k in range(i+1, n):
16            m = arr[k, i]/pivot
17            arr[k, i:] -= m*arr[i, i:]
18    return arr[:, :n], arr[:, n]
19
20 def back_Substitution(A,b):
21     n = A.shape[0]
22     X = np.empty(n)
23     X[-1] = b[-1]/A[-1, -1]
24     for i in range(-2, -(n+1), -1):
25         X[i] = (b[i] - np.dot(A[i, i+1:], X[i+1:]))/A[i, i]
26
27     return X
28
29 def Solve(A,b):
30     """
31     Solve a system S of linear equations  $Ax = b$  defined \
32     by the matrix A and the vector b
33     """
34     a, bb = for_Elimination(A,b)
```

```

35     return back_Substitution(a, bb)
36 if __name__ == "__main__":
37     n = 100
38     A = np.random.randn(n,n)
39
40     b = np.random.randn(n)
41
42     names = ["My method", "numpy method"]
43     t = []
44     # -----
45     t1 =time(); X = Solve(A,b);t.append(time()-t1)
46     t1 =time(); x = np.linalg.solve(A,b);t.append(time()-t1)
47     # -----
48     plt.bar(names,t, label = "n = "+str(n))
49     plt.legend()
50     plt.show()
51     # -----
52     print(list(zip(names,t)))
53     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

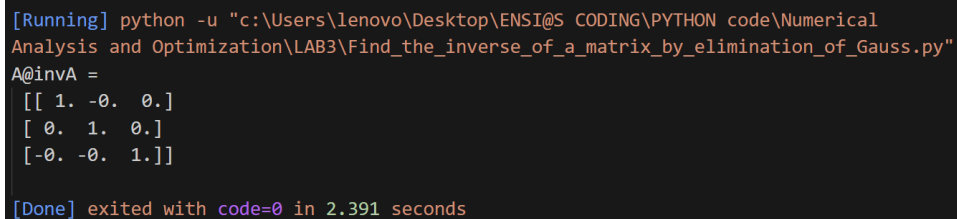
```

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

```
1 from Solve_a_system_by_the_elimination_of_Gauss import Solve
2 import numpy as np
3 def Inverse(A):
4     n = A.shape[0]
5     inv_A = np.empty((n,n))
6     b = np.array([0]*n)
7     for i in range(n):
8         b[i] = 1
9         inv_A[:,i] = Solve(A,b)
10        b[i] = 0
11
12    return inv_A
13 if __name__ == "__main__" :
14     A = np.array([[25,5,1],
15                  [64,8,1],
16                  [144,12,1]])
17
18     invA = Inverse(A)
19     print("A@invA = \n",np.around(A@invA))
```

1.2.1 Screenshots



```
[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\Find_the_inverse_of_a_matrix_by_elimination_of_Gauss.py"
A@invA =
[[ 1. -0.  0.]
 [ 0.  1.  0.]
 [-0. -0.  1.]]
[Done] exited with code=0 in 2.391 seconds
```

1.3 Python implementation for the LU decomposition.

code file : Implementation_for_the_LU_decomposition.py

```
1 import numpy as np
2 import time as t
3 import scipy.linalg
4 np.random.seed(42)
5 # Define the matrix
6 n = 50
7 A = np.random.randn(n,n)
8 # -----| implementation for the LU decomposition|-----
9 def LU(A, ameliorer = False) :
10     n = len(A)
11     U = A.copy()
12     L = np.eye(n)
13     for i in range(n-1):
14
15         pivot = U[i,i]
16         if pivot == 0 : print("pivot = 0") ; return L,U
17
18         for k in range(i+1, n):
19             L[k,i] = U[k,i]/pivot
```

```

20         U[k] = U[k] - L[k,i]*U[i]
21     return L,U
22
23 def PLU(A):
24     n = A.shape[0]
25     U = A.copy()
26     L = np.eye(n, dtype= np.double)
27     P = np.eye(n, dtype= np.double)
28
29     for i in range(n):
30         for k in range(i,n):
31             if ~np.isclose(U[i,i], 0) :
32                 break
33             U[[k,k+1]] = U[[k+1, k]]
34             P[[k,k+1]] = P[[k+1, k]]
35
36         for k in range(i+1, n):
37             L[k,i] = U[k,i]/U[i,i]
38             U[k] = U[k] - L[k,i]*U[i]
39     return P,L,U
40
41
42
43 if __name__ == "__main__":
44     # -----
45     t1 = t.time() ; L,U = LU(A) ;t1 = t.time() - t1
46     t2 = t.time();p, l,u = scipy.linalg.lu(A);t2 = t.time() - t2
47     t3 = t.time();p, pL, pU = PLU(A) ; t3 = t.time() - t3
48     # -----
49     print("my LU time : ",np.around(t1,10))
50     print("my PLU time : ",np.around(t3,10))
51     # -----
52     print("numpy lu time : ",np.around(t2,10))
53     # -----
54     print(f"A is a matrix of shape {n}x{n}")
55     print(f"L == l : {np.sum(np.around(l) == np.around(L))*100/len(l)
56           ↪ **2}%")
57     print(f"U == u : {np.sum(np.around(u) == np.around(U))*100/len(l)
58           ↪ **2}%")
59     print(f"A == L*U : {np.sum(np.around(A) == np.around(L@U))*100/len(
60           ↪ l)**2}%")
61     print(f"A == l*u : {np.sum(np.around(A) == np.around(l@u))*100/len(
62           ↪ l)**2}%")
63     print(f"L@U == l*u : {np.sum(np.around(L@U) == np.around(l@u))*100/
64           ↪ len(l)**2}%")
65
66     print(f"pL == l : {np.sum(np.around(l) == np.around(pL))*100/len(l)
67           ↪ **2}%")
68     print(f"pU == u : {np.sum(np.around(u) == np.around(pU))*100/len(l)
69           ↪ **2}%")
70     print(f"A == pL*pU : {np.sum(np.around(A) == np.around(pL@pU))*100/
71           ↪ len(l)**2}%")
72     print(f"pL@pU == l*u : {np.sum(np.around(pL@pU) == np.around(l@u))
73           ↪ *100/len(l)**2}%")

```

```

[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\Implementation_for_the_LU_decomposition.py"
my LU time : 0.0071957111
my PLU time : 0.0099756718
numpy lu time : 0.0112290382
A is a matrix of shape 50x50
L == l : 63.96%
U == u : 53.12%
A == L*U : 100.0%
A == l*u : 27.84%
L@U == l*u : 27.84%
pL == l : 63.96%
pU == u : 53.12%
A == pL*pU : 100.0%
pL@pU == l*u : 27.84%

[Done] exited with code=0 in 2.184 seconds

```

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

```

1 from Implementation_for_the_LU_decomposition import LU, PLU
2 import numpy as np
3
4 def Solve_Lz_b(L,b):
5     n = L.shape[0]
6     Z = np.empty(n)
7     Z[0] = b[0]/L[0,0]
8     for i in range(1,n):
9         Z[i] = (b[i] - np.dot(L[i,:i], Z[:i]))/L[i,i]
10    return Z
11
12 def Solve_Ux_z(U,z):
13     n = U.shape[0]
14     X = np.empty(n)
15     X[-1] = z[-1]/U[-1,-1]
16     for i in range(-2, -(n+1), -1):
17         X[i] = (z[i] - np.dot(U[i,i+1:], X[i+1:]))/U[i,i]
18    return X
19
20 def Solve(L,U ,b):
21     z = Solve_Lz_b(L,b)
22     x = Solve_Ux_z(U,z)
23    return x
24
25 if __name__ == "__main__":
26     A = np.array([[2,4,-2],
27                   [4,9,-3],
28                   [-2,-3,7]])
29     b = np.array([2,8,10])
30
31     L,U = LU(A)
32     Xlu = Solve(L,U,b)
33     print("x by LU = ",Xlu)
34
35     P,L,U = PLU(A)
36     Xlu = Solve(L,U,np.dot(P,b))
37     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

```
1 import numpy as np
2
3 def Choleski_decomposition(A):
4     L = np.zeros(A.shape, dtype= np.double)
5     n = A.shape[0]
6     for k in range(n):
7         L[k,k] = np.sqrt(A[k,k] - np.dot(L[k,:k], L[k,:k]))
8
9         for i in range(k+1, n):
10             L[i,k] = ( A[i,k] - np.dot(L[i,:k], L[:k,k]) )/L[k,k]
11
12     return L
13 if __name__ == "__main__":
14
15     A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)
16
17
18     L = Choleski_decomposition(A)
19     l = np.linalg.cholesky(A)
20     print("Choleski_decomposition L = \n",L)
21     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

```
1 from Solve_a_system_by_LU_decomposition import Solve_Lz_b as for_sub
2 from Solve_a_system_by_LU_decomposition import Solve_Ux_z as back_sub
3 from Implementation_for_the_LU_decomposition import LU, PLU
4
```

```

5 import numpy as np
6
7 def LU_Inverse(L,U, P = None):
8     n = L.shape[0]
9     Inv = np.empty((n,n), dtype=np.double)
10    x = np.empty(n, dtype=np.double)
11    b = np.zeros(n, dtype=np.double)
12    for i in range(n):
13        b[i] = 1
14        # Solve the equation L * X = b for X using forward substitution
15        if P is not None:
16            x = for_sub(L,np.dot(P,b))
17        else :
18            x = for_sub(L, b)
19        # Solve the equation L^T * Y = X for Y using back substitution
20        x = back_sub(U, x)
21        # Set the i-th column of the inverse matrix
22        Inv[:,i] = x
23        b[i] = 0
24    return Inv
25
26 if __name__ == "__main__":
27     A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)
28
29     L, U = LU(A)
30
31     LU_InvA = LU_Inverse(L,U)
32     P,L,U = PLU(A)
33     PLU_InvA = LU_Inverse(L,U, P)
34     numpy_invA = np.linalg.inv(A)
35     print(f"Inverse of A by LU decomposition is \n {LU_InvA}")
36     print(f"Inverse of A by PLU decomposition is \n {PLU_InvA}")
37     print(f"Inverse of A by numpy is \n {numpy_invA}")

```

1.6.1 Screenshots

```

[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\LU_Inverse.py"
Inverse of A by LU decomposition is
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]
Inverse of A by PLU decomposition is
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]
Inverse of A by numpy is
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]

[Done] exited with code=0 in 0.712 seconds

```

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

code file : Inverse_by_Choleski_decomposition.py

```

1 from Choleski_decomposition import Choleski_decomposition
2 from Solve_a_system_by_LU_decomposition import Solve_Lz_b as for_sub
3 from Solve_a_system_by_LU_decomposition import Solve_Ux_z as back_sub
4 import numpy as np
5
6 def Inverse_by_choleski(L):
7     n = L.shape[0]
8     Inv = np.empty((n,n), dtype=np.double)
9     x = np.empty(n, dtype=np.double)
10    b = np.zeros(n, dtype=np.double)
11    for i in range(n):
12        b[i] = 1
13        # Solve the equation L * X = b for X using forward substitution
14        x = for_sub(L, b)
15        # Solve the equation L^T * Y = X for Y using back substitution
16        x = back_sub(L.T, x)
17        # Set the i-th column of the inverse matrix
18        Inv[:,i] = x
19        b[i] = 0
20    return Inv
21
22    return Inv
23 if __name__ == "__main__":
24     A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)
25
26     L = Choleski_decomposition(A)
27     Choleski_InvA = Inverse_by_choleski(L)
28     numpy_invA = np.linalg.inv(A)
29     print(f"Inverse of A by choleski decomposition is \n {Choleski_InvA
30           ↪ }")
31     print(f"Inverse of A by numpy is \n {numpy_invA}")

```

1.7.1 Screenshots

```

[Running] python -u "c:\Users\lenovo\Desktop\ENSI@S CODING\PYTHON code\Numerical
Analysis and Optimization\LAB3\Inverse_by_choleski_decomposition.py"
Inverse of A by choleski decomposition is
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]
Inverse of A by numpy is
[[0.75 0.5  0.25]
 [0.5  1.   0.5 ]
 [0.25 0.5  0.75]]

[Done] exited with code=0 in 0.73 seconds

```

1.8 Comparing these solving methods

code file : comparing_these_solving_methods.py

```

1 from Find_the_inverse_of_a_matrix_by_elimination_of_Gauss import
   ↪ Inverse as Gauss_Inv
2 from Choleski_decomposition import Choleski_decomposition
3 from Inverse_by_choleski_decomposition import Inverse_by_choleski as
   ↪ Choleski_Inv

```



```

4 from Solve_a_system_by_LU_decomposition import Solve as LU_solve
5 from Solve_a_system_by_the_elimination_of_Gauss import Solve as
  ↳ Gauss_solve
6 from Implementation_for_the_LU_decomposition import LU, PLU
7 from LU_Inverse import LU_Inverse as LU_Inv
8 from time import time
9 import matplotlib.pyplot as plt
10 import numpy as np
11 np.random.seed = 42
12 n = 10
13 names = ["LU_inv", "PLU_inv", "numpy_inv", "Gauss_inv",
14          "Choleski_inv", "LU_solve",
15          "PLU_solve", "Gauss_solve", "numpy_solve"]
16 times = []
17
18 # A = np.array([[2, -1, 0],[-1, 2, -1],[0,-1,2]], dtype= np.double)
19 A = np.random.randn(n,n)
20 b = np.random.randn(n)
21 # -----
22 t = time()
23 l,u = LU(A)
24 LU_Inv(l,u)
25 times.append(time()- t)
26 # -----
27 t = time()
28 p,l,u = PLU(A)
29 LU_Inv(l,u,p)
30 times.append(time()- t)
31 # -----
32 t = time()
33 np.linalg.inv(A)
34 times.append(time()- t)
35 # -----
36 t = time()
37 Gauss_Inv(A)
38 times.append(time()- t)
39 # -----
40 t = time()
41 l = Choleski_decomposition(A)
42 Choleski_Inv(l)
43 times.append(time()- t)
44 # -----
45 t = time()
46 l,u = LU(A)
47 LU_solve(l,u,b)
48 times.append(time()- t)
49 # -----
50 t = time()
51 p,l,u = PLU(A)
52 LU_solve(l,u, np.dot(p,b))
53 times.append(time()- t)
54 # -----
55 t = time()
56 Gauss_solve(A,b)
57 times.append(time()- t)
58 # -----
59 t = time()
60 np.linalg.solve(A,b)

```

```

61 times.append(time()- t)
62 # -----
63
64 plt.bar(names[:-4], times[:-4], alpha = 0.6 )
65 plt.bar(names[-4:], times[-4:], alpha = 0.6 )
66 plt.title("runing time of these method")
67 plt.xlabel("methods names")
68 plt.ylabel("time (s)")
69 plt.show()

```

1.8.1 Screenshots

