CPL_As4

September 13, 2021

[62]:
```
%run My_Library.ipynb
```

[63]:
```
import copy


# Function for partial pivot for LU decomposition

def partial_pivot_LU (mat, vec, n):
    for i in range (n-1):
        if mat[i][i] ==0:
            for j in range (i+1,n):
                # checks for max absolute value and swaps rows
                # of both the input matrix and the vector as well
                if abs(mat[j][i]) > abs(mat[i][i]):
                    mat[i], mat[j] = mat[j], mat[i]
                    vec[i], vec[j] = vec[j], vec[i]
    return mat, vec


# Function to calculate the determinant of a matrix
# via product of transformed L or U matrix

def determinant(mat,n):
    det=1
    for i in range(n):
        det*=-1*mat[i][i]
    return det


# Function to produce n x n identity matrix

def get_identity(n):
    I=[[0 for j in range(n)] for i in range(n)]
    for i in range(n):
        I[i][i]=1
    return I
```

1 Question 1

[64]:
```
# LU decomposition using Doolittle's condition L[i][i]=1
# without making separate L and U matrices

def LU_doolittle(mat,n):
    for i in range(n):
        for j in range(n):
            if i>0 and i<=j: # changing values of upper triangular matrix
                sum=0
                for k in range(i):
                    sum+=mat[i][k]*mat[k][j]
                mat[i][j]=mat[i][j]-sum
            if i>j: # changing values of lower triangular matrix
                sum=0
                for k in range(j):
                    sum+=mat[i][k]*mat[k][j]
                mat[i][j]=(mat[i][j]-sum)/mat[j][j]
    return mat

def for_back_subs_doolittle(mat,n,vect):
    # initialization
    y=[0 for i in range(n)]

    # forward substitution
    y[0]=vect[0]
    for i in range(n):
        sum=0
        for j in range(i):
            sum+=mat[i][j]*y[j]
        y[i]=vect[i]-sum

    # backward substitution
    x[n-1]=y[n-1]/mat[n-1][n-1]
    for i in range(n-1,-1,-1):
        sum=0
        for j in range(i+1,n):
            sum+=mat[i][j]*x[j]
        x[i]=(y[i]-sum)/mat[i][i]
    del(y)
    return x
```

[65]:
```
# LU decomposition using Doolittle's condition L[i][i]=1


print("The matrix is: ")
A1,ro,co = read_matrix('A.txt')
```

```
print_matrix(A1,ro,co)

vector=[6,-3,-2,0]

# partial pivoting to avoid division by zero at pivot place
A1, vector = partial_pivot_LU(A1, vector, ro)
A1 = LU_doolittle(A1,ro)
print("The transformed LU matrix is ")
print_matrix(A1,ro,ro)

x = [0 for i in range(ro)]

x = for_back_subs_doolittle(A1,ro,vector)

print("Solutions are : ")
for i in range(ro):
    print("x["+str(i)+"] = "+str(x[i]))
```

```
The matrix is:

4.0    0.0    1.0    4.0

1.0    3.0    1.0    7.0

3.0    4.0    -4.0    0.0

7.0    -6.0    9.0    2.0


The transformed LU matrix is

4.0    0.0    1.0    4.0

0.25    3.0    0.75    6.0

0.75    1.3333333333333333    -5.75    -11.0

1.75    -2.0    -1.5217391304347827    -9.73913043478261


Solutions are :
x[0] = 0.3928571428571428
x[1] = -4.6875
x[2] = -3.8928571428571423
x[3] = 2.080357142857143
```

```
[ ]:
```

[66]:
```
# LU decomposition using Crout's condition U[i][i]=1

def LU_crout(mat,n):
    for i in range(n):
        for j in range(n):
            if i>=j: # changing values of lower triangular matrix
                sum=0
                for k in range(j):
                    sum+=mat[i][k]*mat[k][j]
                mat[i][j]=mat[i][j]-sum
            if i<j: # changing values of uppr triangular matrix
                sum=0
                for k in range(i):
                    sum+=mat[i][k]*mat[k][j]
                mat[i][j]=(mat[i][j]-sum)/mat[i][i]
    return mat

def for_back_subs_crout(mat,n,vect):
    y=[0 for i in range(n)]

    # forward substitution
    y[0]=vect[0]/mat[0][0]
    for i in range(n):
        sum=0
        for j in range(i):
            sum+=mat[i][j]*y[j]
        y[i]=(vect[i]-sum)/mat[i][i]

    # backward substitution
    x[n-1]=y[n-1]
    for i in range(n-1,-1,-1):
        sum=0
        for j in range(i+1,n):
            sum+=mat[i][j]*x[j]
        x[i]=y[i]-sum
    del(y)
    return x
```

[67]:
```
# LU decomposition using Crout's condition U[i][i]=1

print("The matrix is: ")
A2,ro,co=read_matrix('A.txt')
print_matrix(A2,ro,co)

vector=[6,-3,-2,0]

# partial pivoting to avoid division by zero at pivot place
```

```
A1, vector = partial_pivot_LU(A1, vector, ro)
A2=LU_crout(A2,ro)
print("The transformed LU matrix is ")
print_matrix(A2,ro,ro)

x = [0 for i in range(ro)]

x=for_back_subs_crout(A2,ro,vector)

print("Solutions are : ")
for i in range(ro):
    print("x["+str(i)+"] = "+str(x[i]))
```

```
The matrix is:

4.0    0.0    1.0    4.0

1.0    3.0    1.0    7.0

3.0    4.0    -4.0    0.0

7.0    -6.0    9.0    2.0


The transformed LU matrix is

4.0    0.0    0.25    1.0

1.0    3.0    0.25    2.0

3.0    4.0    -5.75    1.9130434782608696

7.0    -6.0    8.75    -9.73913043478261


Solutions are :
x[0] = 0.3928571428571428
x[1] = -4.6875
x[2] = -3.892857142857143
x[3] = 2.080357142857143
```

2 Question 2

[68]:
```
def inverse_by_lu_decomposition (matrix, n):

    identity=get_identity(ro)
```

```
x=[]

'''
The inverse finding process could have been done using
a loop for the four columns. But while partial pivoting,
the rows of final inverse matrix and the vector both are
also interchanged. So it is done manually for each row and vector.

deepcopy() is used so that the original matrix doesn't change on
changing the copied entities. We reuire the original multiple times here

1. First the matrix is deepcopied.
2. Then partial pivoting is done for both matrix and vector.
3. Then the decomposition algorithm is applied.
4. Then solution is obtained.
5. And finally it is appended to a separate matrix to get the inverse.
Note: The final answer is also deepcopied because there is some error
    due to which all x0, x1, x2 and x3 are also getting falsely appended.
'''

matrix_0 = copy.deepcopy(matrix)
partial_pivot_LU(matrix_0, identity[0], n)
matrix_0 = LU_doolittle(matrix_0, n)
x0 = for_back_subs_doolittle(matrix_0, n, identity[0])
x.append(copy.deepcopy(x0))


matrix_1 = copy.deepcopy(matrix)
partial_pivot_LU(matrix_1, identity[1], n)
matrix_1 = LU_doolittle(matrix_1, n)
x1 = for_back_subs_doolittle(matrix_1, n, identity[1])
x.append(copy.deepcopy(x1))

matrix_2 = copy.deepcopy(matrix)
partial_pivot_LU(matrix_2, identity[2], n)
matrix_2 = LU_doolittle(matrix_2, n)
x2 = for_back_subs_doolittle(matrix_2, n, identity[2])
x.append(copy.deepcopy(x2))

matrix_3 = copy.deepcopy(matrix)
partial_pivot_LU(matrix_3, identity[3], n)
matrix_3 = LU_doolittle(matrix_3, n)
x3 = for_back_subs_doolittle(matrix_3, n, identity[3])
x.append(copy.deepcopy(x3))

# The x matrix to be transposed to get the inverse in desired form
inverse,r,c=transpose_matrix(x,n,n)
```

```
    return (inverse)
```

[69]:
```
print("The initial matrix is : ")
B,ro,co=read_matrix('B.txt')
print_matrix(B,ro,ro)

C=copy.deepcopy(B) # deepcopy for unchanged matrix required for inverse

identity=get_identity(ro)

# Then partial pivoting is done for both matrix and vector.
# Then the decomposition algorithm is applied.
B, identity = partial_pivot_LU(B, identity, ro)
B=LU_doolittle(B,ro)

#print("The transformed LU matrix is ")
#print_matrix(B,ro,ro)

#Checking if inverse exists
det=determinant(B,ro)
if det == 0:
    print("Determinant = zero.\nInverse doesn't exist.")
else:
    print("The inverse is:")

    # Calculating and printing inverse
    inverse= inverse_by_lu_decomposition(C, ro)
    print_matrix(inverse,ro,ro)

    # Verification: gives indentity matrix on multiplication with original matrix
    print("Verification : ")
    mm,r,c=matrix_multiply(C,ro,ro,inverse,ro,ro)
    print_matrix(round_matrix(mm),r,c)
```

```
The initial matrix is :

4.0    0.0    3.0    3.0

0.0    1.0    4.0    3.0

1.0    2.0    4.0    9.0

2.0    5.0    0.0    0.0


The inverse is:
```

```
0.23668639053254437     -0.14792899408284024     -0.02958579881656806
0.04142011834319527

-0.09467455621301776     0.059171597633136064     0.011834319526627168
0.18343195266272194

0.04142011834319527     0.34911242603550297     -0.1301775147928994
-0.017751479289940832

-0.023668639053254437     -0.15187376725838264     0.16962524654832348
-0.037475345167652864


Verification :

1.0     0     0     -0.0

0     1.0     0     0.0

-0.0     -0.0     1.0     0

-0.0     -0.0     -0.0     1.0
```

3 Question 3

[70]:
```
# Function for Cholesky decomposition
# Only works for Hermitian and positive definite matrices

def LU_cho(mat,n):
    for i in range(n):
        for j in range(i,n):
            if i==j: # changing diagonal elements
                sum=0
                for k in range(i):
                    sum+=mat[i][k]**2
                mat[i][i]=math.sqrt(mat[i][i]-sum)
            if i<j: # changing upper traiangular matrix
                sum=0
                for k in range(i):
                    sum+=mat[i][k]*mat[k][j]
                mat[i][j]=(mat[i][j]-sum)/mat[i][i]

                # setting the lower triangular elements same as elements at the␣
 ↪transposition
```

```
                mat[j][i]=mat[i][j]
    return mat

def for_back_subs_cho(mat,n,vect):
    y=[0 for i in range(n)]

    # forward substitution
    y[0]=vect[0]/mat[0][0]
    for i in range(n):
        sum=0
        for j in range(i):
            sum+=mat[i][j]*y[j]
        y[i]=(vect[i]-sum)/mat[i][i]

    # forward substitution
    x[n-1]=y[n-1]
    for i in range(n-1,-1,-1):
        sum=0
        for j in range(i+1,n):
            sum+=mat[i][j]*x[j]
        x[i]=(y[i]-sum)/mat[i][i]
    del(y)
    return x
```

[71]:
```
# Function for Cholesky decomposition

print("The matrix is: ")
C,ro,co=read_matrix('As4matrixC.txt')
print_matrix(C,ro,co)

vector=[2.20, 2.85, 2.79, 2.87]

# partial pivoting to avoid division by zero at pivot place
C, vector = partial_pivot_LU(C, vector, ro)
C=LU_cho(C,ro)
print("The transformed Cholesky matrix is ")
round_matrix(C)
print_matrix(C,ro,ro)

x=for_back_subs_cho(C,ro,vector)

print("Solutions are : ")
for i in range(ro):
    print('%.2f'%x[i])
```

```
The matrix is:
```

```
10.2    1.2    0.0    2.6    2.86

1.8    12.6    -0.4    1.2    2.87

0.0    -0.5    9.7    0.0    2.88

3.7    1.7    0.0    6.2    2.98


The transformed Cholesky matrix is

3.19    0.38    0    0.81

0.38    3.53    -0.11    0.25

0    -0.11    3.11    0.01

0.81    0.25    0.01    2.34


Solutions are :
0.10
0.19
0.30
0.39
```

[72]:
```python
# LU decomposition using Doolittle's condition L[i][i]=1
# by making separate L and U matrices

def LU_do2(M,n):
    # initialization
    L=[[0 for j in range(n)] for i in range(n)]
    U=[[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        L[i][i]=1
        for j in range(n):
            if i>j:
                U[i][j]=0
            elif i<j:
                L[i][j]=0
            U[0][j]=M[0][j]
            L[i][0]=M[i][0]/U[0][0]
            if i>0 and i<=j: # changing values for upper traiangular matrix
                sum=0
                for k in range(i):
                    sum+=L[i][k]*U[k][j]
```

```
                U[i][j]=M[i][j]-sum
            if i>j: # changing values for lower traiangular matrix
                sum=0
                for k in range(j):
                    sum+=L[i][k]*U[k][j]
                L[i][j]=(M[i][j]-sum)/U[j][j]
    print_matrix(L,n,n)
    print_matrix(U,n,n)

    # To check if the L and U matrices are correct, use this for verification
    m,r,c=matrix_multiply(L,ro,ro,U,ro,ro)
    print_matrix(m,r,c)

    return M
```

[ ]: