+ Code + Text

# Algorithm for finding determinant

```
Input: A - square matrix of size n x n in form of numpy array.
Output: det - determinant of A

if n equals 1, return A[0,0]
otherwise, initialize det to 0
for each column j of the first row:
a. get the submatrix obtained by deleting row 0 and column j
b. recursively compute the determinant of the submatrix using step 1-3
c. multiply the determinant by (-1)^j and A[0,j]
d. add the product to det
return det
```

```
import numpy as np
def determinant(A):
    n = A.shape[0]
    if n == 1:
        return A[0, 0]
    else:
        det = 0
        for j in range(n):
            # Get submatrix obtained by deleting row 0 and column j
            submatrix = np.delete(np.delete(A, 0, axis=0), j, axis=1)
            # Compute determinant recursively
            det += (-1) ** j * A[0, j] * determinant(submatrix)
        return det
```

# Algorithm for finding LU Decomposition(Do Little) of A

```
1. Define the function with input matrix A
2. Check if the determinant of A is zero. If yes, print error message and return None.
3. Initialize n as the length of A and create an identity matrix P of size n to store the permutation matrix.
4. Create an identity matrix L of size n to store the lower triangular matrix.
5. Create a zero matrix U of size n to store the upper triangular matrix.
6. For each column k in the range 0 to n-1, do the following:
   a. Perform partial pivoting by swapping rows to ensure diagonal element is the largest absolute value in the column.
   b. Update the permutation matrix P based on the row swaps.
   c. For each row i in the range k+1 to n-1, do the following:
      i. Compute the ratio lam = A[i,k] / A[k,k]
      ii. Set L[i,k] = lam
      iii. Update the remaining entries of row i of A using A[i,k+1:n] = A[i,k+1:n] - lam * A[k,k+1:n]
7. Set U as the upper triangular part of A.
8. Return L, U, P, and A as the output of the function.
```

```
import numpy as np


def LUdecomp(A):
    if determinant(A)==0:
      print("Matrix is Singular LU decomposition not possible")
      return None
    else:
     n = len(A)
     P = np.eye(n)  # initialize permutation matrix as identity matrix
     L = np.eye(n)
     U = np.zeros(n)
     for k in range(0, n-1):
        # Partial pivoting: swap rows to ensure diagonal element is largest absolute value in the column
        max_index = k + np.argmax(np.abs(a[k:, k]))
        if max_index > k:
            A[[k, max_index]] = A[[max_index, k]]
            P[[k, max_index]] = P[[max_index, k]]
        for i in range(k+1, n):
```

```
            lam = A[i,k]/A[k,k]
            L[i,k] = lam
            A[i,k+1:n] = A[i,k+1:n] - lam*A[k,k+1:n]
            A[i,k] = 0

    U=np.triu(A)
    print('LU decomposition of A is given by \n L=\n',np.matrix(L),'and \n U=\n',np.matrix(U),"by using permutation matrix \n P=\n",np.matrix
    return
```

## Examples:

```
a = np.array([[0, 1, -1],
              [2, -4, 9],
              [1, -2, 1]],dtype=float)

LUdecomp(a)
```

```
LU decomposition of A is given by
 L=
 [[1.  0.  0. ]
 [0.  1.  0. ]
 [0.5 0.  1. ]] and
 U=
 [[ 2.  -4.   9. ]
 [ 0.   1.  -1. ]
 [ 0.   0.  -3.5]] by using permutation matrix
 P=
 [[0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

```
a = np.array([[0, 1, -1],
              [2, -4, 9],
              [0, 2, -2]],dtype=float)

LUdecomp(a)
```

```
Matrix is Singular LU decomposition not possible
```

## Python Library function for getting LU decomposition

```
a = np.array([[0, 1, -1],
              [2, -4, 9],
              [1, -2, 1]])

from scipy.linalg import lu

P, L, U = lu(a)
print('A:\n', a)
print('P:\n', P)
print('L:\n', L)
print('U:\n', U)
print('LU:\n',np.dot(L, U))
```

```
A:
 [[ 0  1 -1]
 [ 2 -4  9]
 [ 1 -2  1]]
P:
 [[0.000 1.000 0.000]
 [1.000 0.000 0.000]
 [0.000 0.000 1.000]]
L:
 [[1.000 0.000 0.000]
 [0.000 1.000 0.000]
 [0.500 0.000 1.000]]
U:
 [[2.000 -4.000 9.000]
```

```
 [0.000 1.000 -1.000]
 [0.000 0.000 -3.500]]
LU:
 [[2.000 -4.000 9.000]
 [0.000 1.000 -1.000]
 [1.000 -2.000 1.000]]
```

## Crout's Decomposition

```
import numpy as np

def crout(A):
  if determinant(A)==0:
      print("Matrix is Singular LU decomposition not possible")
      return None
  else:
    n = len(A)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for j in range(n):
        U[j, j] = 1.0
        for i in range(j, n):
            s1 = sum(U[k, j] * L[i, k] for k in range(j))
            L[i, j] = A[i, j] - s1

        for i in range(j+1, n):
            s2 = sum(U[k, i] * L[j, k] for k in range(j))
            if L[j,j]==0:
               print("Crout decomposition not possible without row exchange")
               return None
            else:
              U[j, i] = (A[j, i] - s2) / L[j, j]
    print('LU decomposition of A is given by \n L=\n',np.matrix(L),'and \n U=\n',np.matrix(U))
    return L, U

# Define matrix
A = np.array([[0.52,0.2,0.25],
              [0.3,0.5,0.2],
              [0.18,0.3,0.55]],dtype=float)

# Perform Crout's decomposition
L,U=crout(A)
```

```
LU decomposition of A is given by
 L=
 [[0.52       0.         0.        ]
 [0.3        0.38461538 0.        ]
 [0.18       0.23076923 0.43      ]] and
 U=
 [[1.         0.38461538 0.48076923]
 [0.         1.         0.145     ]
 [0.         0.         1.        ]]
```

## Verification of Decomposition

```
np.dot(L,U)
```

```
array([[ 4.,  3., -5.],
       [-2., -4., -5.],
       [ 8.,  8.,  0.]])
```

```
##GaussSiedel Algorithm

```

Input:
   - A: an n by n coefficient matrix
   - b: an n by 1 matrix
   - x: an n by 1 matrix representing the initial guess for the solution
```
```

```
    - x: an n by 1 matrix representing the initial guess for the solution
    - tol: the tolerance for convergence
    - N: the maximum number of iterations
    -Note give all matrices in form of numpy arrays.



1. Start by initializing the iteration count k to 0 and the convergence flag
converged to False.
2. If any diagonal entry of the coefficient matrix A is zero, swap rows to move
a non-zero entry to the diagonal.
3. Enter a while loop that continues until either convergence is achieved or
the maximum number of iterations N is reached.
4. Inside the while loop, increment the iteration count k by 1 and make a copy
of the solution vector x as x0.
5. Use a nested loop to compute the updated value of each component of x using
the Gauss-Seidel iteration formula.
6. Check whether the difference between x and x0 is less than the tolerance tol
using the norm function from numpy.linalg. If the condition is satisfied, set
the convergence flag converged to True and print the message "Converged!". Exit
the while loop using the break statement.
7. If convergence has not been achieved after N iterations, print the message
"Not converged, increase the # of iterations".
8. Print either converged or not converged.


```
```

---

## GaussSiedel Algorithm

```
Input:
   - A: an n by n coefficient matrix
   - b: an n by 1 matrix
   - x: an n by 1 matrix representing the initial guess for the solution
   - tol: the tolerance for convergence
   - N: the maximum number of iterations
   -Note give all matrices in form of numpy arrays.



1. Start by initializing the iteration count k to 0 and the convergence flag converged to False.
2. If any diagonal entry of the coefficient matrix A is zero, swap rows to move a non-zero entry to the diagonal.
3. Enter a while loop that continues until either convergence is achieved or the maximum number of iterations N is reached.
4. Inside the while loop, increment the iteration count k by 1 and make a copy of the solution vector x as x0.
5. Use a nested loop to compute the updated value of each component of x using the Gauss-Seidel iteration formula.
6. Check whether the difference between x and x0 is less than the tolerance tol using the norm function from numpy.linalg. If the condition is sat
7. If convergence has not been achieved after N iterations, print the message "Not converged, increase the # of iterations".
8. Print either converged or not converged.
```

```python
import numpy as np
import numpy.linalg as LA
from tabulate import tabulate

def Gauss_Siedel(A, b, x, tol, N):

    k = 0
    data=[]
    converged = False
    for i in range(len(b)):
        if A[i,i] == 0:
            j = i + np.argmax(np.abs(A[i:, i]))
            A[[i, j]] = A[[j, i]]
            b[[i, j]] = b[[j, i]]

    while k <= N:
     k += 1
     x0 = x.copy()
     data.append([k,x0[0],x0[1],x0[2]])
     for i in range(len(b)):
```
```

```
        Sum = 0  # reset Sum to zero before the inner loop
        for j in range(len(b)):
            if j != i:
                Sum += A[i,j] * x[j]
        x[i] = (1 / A[i,i]) * (b[i] - Sum)

        if LA.norm(x - x0) < tol:
            converged = True
            print(tabulate(data,headers=['Iter no','x1','x2',"x3"],tablefmt="github"))
            print('Converged!in',k,'no of iterations')
            return None  # exit the while loop if converged

    if not converged:
      print(tabulate(data,headers=['Iter no','x1','x2',"x3"],tablefmt="github"))
      print('Not converged, increase the # of iterations')

      return
```

## Examples

```
A=np.array([[8,8,3],[2,8,5],[3,5,10]],dtype=float)

b=np.array([[14],[5],[-8]],dtype=float)
N=10
tol=0.1
x=np.zeros_like(b)
Gauss_Siedel(A,b,x,tol,N)
```

```
|   Iter no |       x1 |       x2 |       x3 |
|-----------|----------|----------|----------|
|         1 | 0        | 0        |  0       |
|         2 | 1.75     | 0.1875   | -1.41875 |
|         3 | 2.09453  | 0.988086 | -1.9224  |
|         4 | 1.48281  | 1.4558   | -1.97274 |
|         5 | 1.03398  | 1.59947  | -1.90993 |
|         6 | 0.866754 | 1.60202  | -1.86103 |
Converged!in 6 no of iterations
```

```
A = np.array([[0, 1, -1],
              [2, -4, 9],
              [0, 2, -2]],dtype=float)
b=np.array([[14],[5],[-8]],dtype=float)
N=100
tol=0.1
x=np.zeros_like(b)
Gauss_Siedel(A,b,x,tol,N)
```

```
|        73 | -3200.5 | 1292 | 1296 |
|        74 | -3245.5 | 1310 | 1314 |
|        75 | -3290.5 | 1328 | 1332 |
|        76 | -3335.5 | 1346 | 1350 |
|        77 | -3380.5 | 1364 | 1368 |
|        78 | -3425.5 | 1382 | 1386 |
|        79 | -3470.5 | 1400 | 1404 |
|        80 | -3515.5 | 1418 | 1422 |
|        81 | -3560.5 | 1436 | 1440 |
|        82 | -3605.5 | 1454 | 1458 |
|        83 | -3650.5 | 1472 | 1476 |
|        84 | -3695.5 | 1490 | 1494 |
|        85 | -3740.5 | 1508 | 1512 |
|        86 | -3785.5 | 1526 | 1530 |
|        87 | -3830.5 | 1544 | 1548 |
|        88 | -3875.5 | 1562 | 1566 |
|        89 | -3920.5 | 1580 | 1584 |
|        90 | -3965.5 | 1598 | 1602 |
|        91 | -4010.5 | 1616 | 1620 |
|        92 | -4055.5 | 1634 | 1638 |
|        93 | -4100.5 | 1652 | 1656 |
|        94 | -4145.5 | 1670 | 1674 |
|        95 | -4190.5 | 1688 | 1692 |
|        96 | -4235.5 | 1706 | 1710 |
|        97 | -4280.5 | 1724 | 1728 |
|        98 | -4325.5 | 1742 | 1746 |
|        99 | -4370.5 | 1760 | 1764 |
|       100 | -4415.5 | 1778 | 1782 |
|       101 | -4460.5 | 1796 | 1800 |
Not converged, increase the # of iterations
```



0s completed at 1:52 PM