UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI - UFSJ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA - DEPEL
COORDENAÇÃO DE ENGENHARIA ELÉTRICA - COELE

GABRIEL AUGUSTO SILVA BATISTA

TRABALHO 2

São João del-Rei – MG
Setembro de 2023

Fatoração LU com pivoteamento parcial

```python
import numpy as np

def LU(A, B):
    n = len(A)
    U = np.copy(A)
    L = np.eye(n)
    P = np.eye(n)

    for k in range(n - 1):
        lp = np.argmax(np.abs(U[k:, k])) + k
        if lp != k:
            U[[k, lp]] = U[[lp, k]]
            P[[k, lp]] = P[[lp, k]]
            if k > 0:
                L[[k, lp], :k] = L[[lp, k], :k]

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

```

Nesse código foi utilizado a biblioteca numpy para facilitar as operações.

Na linha 3 defini uma função para realizar a fatorações LU com pivoteamento parcial, que tem como objetivo organizar as linhas da matriz para tornar os cálculos mais estáveis.

O laço de repetição for k in range(n - 1): itera pelas colunas da matriz A, começando da primeira coluna até a penúltima. Esse loop é responsável por realizar a decomposição LU com pivoteamento parcial.

lp = np.argmax(np.abs(U[k:, k])) + k encontra o índice da linha com o maior valor absoluto na coluna atual (coluna k) da matriz U. Isso é chamado de pivoteamento parcial, onde é procurado a melhor linha para ser a linha pivô.

Se o índice da melhor linha (lp) for diferente do índice atual (k), significa que precisamos trocar as linhas k e lp nas matrizes U e P. Isso tem como objetivo evitar problemas de divisão por zero ou números muito pequenos.

O código também verifica se estamos além da primeira coluna (k > 0). Se sim, ele faz uma troca similar nas primeiras colunas da matriz L para garantir que L seja construída corretamente após as trocas de linhas.

O laço de repetição for i in range(k + 1, n): itera pelas linhas abaixo da linha pivô atual (k). Ele calcula os elementos da matriz L e atualiza a matriz U de acordo com a decomposição LU.

m = U[i, k] / U[k, k] calcula o fator multiplicador (m) usado para transformar a matriz U em uma forma triangular superior.

L[i, k] = m armazena o valor de m na matriz L.

U[i, k:] -= m * U[k, k:] realiza a eliminação gaussiana na matriz U para tornar seus elementos abaixo da linha pivô iguais a zero.

No final desses loops, você terá a matriz A decomposta em duas partes: L (matriz triangular inferior) e U (matriz triangular superior), e a matriz de permutação P (que registra as trocas de linhas). Isso permite que o sistema de equações seja resolvido de forma eficiente usando substituição progressiva e retrocedendo para encontrar as variáveis desconhecidas.

```
22      # Triangular inferior, encontra Y
23      Pb = np.dot(P, B)
24      Y = np.zeros(n)
25      for i in range(n):
26          Y[i] = Pb[i] - np.sum(L[i, :i] * Y[:i])
27
28      # Triangular superior, encontra X
29      X = np.zeros(n)
30      for i in range(n - 1, -1, -1):
31          X[i] = (Y[i] - np.sum(U[i, i+1:] * X[i+1:])) / U[i, i]
32
33      return X
```

Nessa parte ainda dentro da função LU, primeiro encontro Y usando a matriz L e o vetor B, em seguida, encontro X usando a matriz U e os valores de Y. A função retorna X que é a solução do sistema.

```python

# Primeira matriz

A = np.array([[3, -4, 1],[1, 2, 2],[4, 0, -3]], dtype='double')
B = np.array([9, 3, -2], dtype='double')
X = LU(A, B)
Ax = np.dot(A, X)
dif = B - Ax

print(f'Resultado da primeira matriz: {X}\nDiferença: {dif}\n')

# Segunda matriz

A2 = np.array([[0.1, -3, 4, 7, 4, 14],[-2, 4, 2, 5, -5, 21],[1, 200, 3, -3, 3, -4],[-1, 5, 4, 22, 7, -8],[4, 8, 7, 10, -9,
B2 = np.array([14, 26, 19, -5, 7, -4], dtype='double')
X2 = LU(A2, B2)
Ax2 = np.dot(A2, X2)
dif2 = B2 - Ax2

print(f'Resultado da segunda matriz: {X2}\nDiferença: {dif2}\n')
```

Aqui é definido as matrizes e seus valores são passados para a função LU que executa o processo demonstrado anteriormente, no Ax é usado a função np.dot para calcular o produto entre as matrizes A e X, após isso é realizado o calculo da diferença entre B e Ax, a mesma logica é aplicada para as duas matrizes.

Resultado:

```
Resultado da primeira matriz: [ 1. -1.  2.]
Diferença: [0. 0. 0.]

Resultado da segunda matriz: [-0.11139017  0.12961284 -0.17011024  0.2833979  -0.41245904  1.05331924]
Diferença: [-7.10542736e-15  0.00000000e+00 -3.55271368e-15 -1.77635684e-15
 -8.88178420e-16 -4.44089210e-16]
```

Esses foram os resultados encontrados após a execução do programa.