Cálculo Numérico

Trabalho 2

Aline Werner

11 de maio de 2018

Introdução

Este trabalho tem como objetivo colocar em prática os conceitos aprendidos durante as aulas de Cálculo Numérico a respeito de sistemas lineares e zeros de funções através da programação na linguagem Python.

Questão 1

Pergunta: Use um programa em Python para resolver o sistema abaixo por decomposição LU.

$$\begin{cases} 2x_1 + 3x_2 + x_3 + 5x_4 = 11 \\ x_1 + 3.5x_2 + x_3 + 7.5x_4 = 13 \\ 1.4x_1 + 2.7x_2 + 5.5x_3 + 12x_4 = 21.6 \\ -2x_1 + 1x_2 + 3x_3 + 28x_4 = 30 \end{cases}$$

Além da solução do sistema, mostre também as matrizes L e U obtidas.

Resposta: Para a resolução do sistema linear acima, foi utilizado o programa mostrado na Figura 1. À direita, é mostrada a função resposável por decompor a matriz em um produto de matrizes L e U. À esquerda, são mostradas as funções responsáveis por resolver o sistema triangular inferior e, em seguida, o superior. Na Figura 2, são mostrados os resultados obtidos: as matrizes L e U e o vetor x de soluções.

Os valores encontrados foram: $x_1 = x_2 = x_3 = x_4 = 1$.

```python
import numpy as np

def decomposicaoLU(A):

    L = []
    U = []

    for i in range(n):
        L.append([0]*n)
        U.append([0]*n)

    for m in range(n):
        L[m][m] = 1

        soma = 0
        for k in range(n):
            soma = soma + L[m][k]*U[k][m]
        U[m][m] = A[m][m] - soma

        for j in range(m,n):
            soma = 0
            for k in range(m):
                soma =  soma + L[m][k]*U[k][j]
            U[m][j] = A[m][j] - soma

        for i in range(m+1,n):
            soma = 0
            for k in range(m+1):
                soma = soma +  L[i][k]*U[k][m]
            L[i][m] = (A[i][m] - soma)/U[m][m]

    print('U = \n'+str(np.matrix(U))+'\n')
    print('L = \n'+str(np.matrix(L))+'\n')

    return L, U
```

```python
def inferior(L, b):

    y1 = b[0]/L[0][0]
    y = [y1]

    for i in range(1,len(L)):
        soma = 0
        for j in range(0,i):
            soma = soma +  L[i][j]*y[j]
        y.append((b[i] - soma)/L[i][i])

    print('y = '+str(np.matrix(y))+'\n')

    return y

def superior(U, y):
    n = len(U)-1
    xn = y[n]/U[n][n]

    x = [0]*len(U)
    x[-1] = xn

    for i in range(n-1,-1,-1):
        soma = 0
        for j in range(i+1,n+1):
            soma = soma +  U[i][j]*x[j]
        xi = (y[i]-soma)/U[i][i]
        x[i] = xi

    print('x = '+str(np.matrix(x)))

    return x



A = [[2.0,  3.0,  1.0,  5.0],
     [1.0,  3.5,  1.0,  7.5],
     [1.4,  2.7,  5.5,  12 ],
     [-2.0, 1.0,  3.0,  28 ]]

b = [11, 13, 21.6, 30]

n = 4

decomposicaoLU(A)
y = inferior(L, b)
superior(U, y)
```

Figura 1: Programa queresolve um sistema linear por decomposição LU

```
U =
[[ 2.          3.          1.          5.        ]
 [ 0.          2.          0.5         5.        ]
 [ 0.          0.          4.65        7.        ]
 [ 0.          0.          0.         18.48387097]]

L =
[[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.7         0.3         1.          0.        ]
 [-1.          2.          0.64516129  1.        ]]

y = [[11.          7.5        11.65       18.48387097]]

x = [[1. 1. 1. 1.]]
```

Figura 2: Resultados obtidos

Questão 2

Pergunta: Desenvolva um programa para resolver o sistema abaixo, utilizando algum método interativo.

$$\begin{cases} 6.1x_1 + 0.32x_2 + 1.3x_3 + 2.1x_4 + 0.11x_5 = 19.52 \\ 0.82x_1 + 8.81x_2 + 1.01x_3 + 3x_4 + 3.12x_5 = 15.83 \\ 0.5x_1 + 1.78x_2 + 15.2x_3 + 4.2x_4 + 8.1x_5 = -22.14 \\ 4.2x_1 + 5.3x_2 + 1.8x_3 + 20.9x_4 + 7.51x_5 = 27.28 \\ 0.2x_1 + 9.1x_2 + 4.68x_3 + 4.3x_4 + 20.1x_5 = -21.78 \end{cases}$$

Depois disso, use os programas apresentados em aula para resolver esse mesmo sistema por eliminação de Gauss e compare os resultados.

Resposta: O programa utilizado para resolver o sistema linear acima é mostrado na Figura 3, e os resultados obtidos com ele nas últimas iterações na Figura 4. O método iterativo utilizado foi o de Jacobi-Richardson e foram realizadas 25 iterações.

```
x = [0,0,0,0,0]

x1 = lambda x2, x3, x4, x5: (19.52-0.32*x2-1.3*x3-2.1*x4-0.11*x5)/6.1
x2 = lambda x1, x3, x4, x5: (15.83-0.82*x1-1.01*x3-3*x4-3.12*x5)/8.81
x3 = lambda x1, x2, x4, x5: (-22.14-0.5*x1-1.78*x2-4.2*x4-8.1*x5)/15.2
x4 = lambda x1, x2, x3, x5: (27.28-4.2*x1-5.3*x2-1.8*x3-7.51*x5)/20.9
x5 = lambda x1, x2, x3, x4: (-21.78-0.2*x1-9.1*x2-4.68*x3-4.3*x4)/20.1

for k in range(25):
    x = [x1(x[1],x[2],x[3],x[4]),
         x2(x[0],x[2],x[3],x[4]),
         x3(x[0],x[1],x[3],x[4]),
         x4(x[0],x[1],x[2],x[4]),
         x5(x[0],x[1],x[2],x[3])]

    print("x"+str(k+1)+" = (%.4f,"%x[0],"%.4f,"%x[1],"%.4f)"%x[2],
                          "%.4f)"%x[3],"%.4f)"%x[4])
```

Figura 3: Programa que resolve um sistema linear pelo método de Jacobi-Richardson

```
x15 = (3.0322, 2.0450, -0.9514) 1.0437) -1.9535)
x16 = (2.9714, 1.9601, -1.0432) 0.9612) -2.0413)
x17 = (3.0254, 2.0355, -0.9616) 1.0344) -1.9633)
x18 = (2.9774, 1.9685, -1.0341) 0.9694) -2.0326)
x19 = (3.0200, 2.0280, -0.9697) 1.0272) -1.9710)
x20 = (2.9822, 1.9752, -1.0269) 0.9759) -2.0257)
x21 = (3.0158, 2.0221, -0.9761) 1.0214) -1.9772)
x22 = (2.9860, 1.9804, -1.0212) 0.9810) -2.0203)
x23 = (3.0125, 2.0174, -0.9812) 1.0169) -1.9820)
x24 = (2.9889, 1.9845, -1.0167) 0.9850) -2.0160)
x25 = (3.0098, 2.0137, -0.9851) 1.0133) -1.9858)
```

Figura 4: Resultados obtidos

Os resultados obtidos com o programas apresentados em aula para resolver esse mesmo sistema por eliminação de Gauss chegou a resultados parecidos, como mostrado na Figura 5, porém com bem menos iterações.

```
(3.2000, 1.4990, -1.7374, 0.4317, -1.4819,) err= 1.0
(3.3697, 2.0602, -1.1383, 0.7362, -1.9423,) err= 0.1777908460238031
(3.1161, 2.0744, -0.9704, 0.9345, -2.0277,) err= 0.08139515715528173
(3.0128, 2.0275, -0.9708, 0.9979, -2.0189,) err= 0.03427295163141849
(2.9934, 2.0047, -0.9897, 1.0061, -2.0058,) err= 0.007631508845839686
(2.9956, 1.9992, -0.9984, 1.0030, -2.0006,) err= 0.002908262740305721
(2.9987, 1.9991, -1.0004, 1.0007, -1.9997,) err= 0.001032382581137342
(2.9999, 1.9997, -1.0003, 1.0000, -1.9998,) err= 0.0003986746915605137
```

Figura 5: Resultados obtidos com eliminação de Gauss

Questão 3

Pergunta: Sistemas massa-mola idealizados têm numerosas aplicações em toda a engenharia. A figura abaixo mostra um arranjo de quatro molas em série sendo comprimidas por uma força de 2000kg. No equilíbrio, as equações de balanço das forças podem ser reduzidas escrevendo-se as interações entre as molas:

$$\begin{cases} k_2(x_2 - x_1) = k_1 x_1 \\ k_3(x_3 - x_2) = k_2(x_2 - x_1) \\ k_4(x_4 - x_3) = k_3(x_3 - x_2) \\ F = k_4(x_4 - x_3) \end{cases}$$

onde os k são as constantes elásticas das molas. Se k_1 até k_4 forem 150, 50, 75 e $225 N/m$, respectivamente, calcule os x.

Resposta: Substituindo os valores e rearrumando os termos, chegamos ao seguinte sistema:

$$\begin{cases} -4x_1 + x_2 + 0x_3 + 0x_4 = 0 \\ 50x_1 - 125x_2 + 75x_3 + 0x_4 = 0 \\ 0x_1 + 75x_2 - 300x_3 + 225x_4 = 0 \\ 0x_1 + 0x_2 - 225x_3 + 225x_4 = 2000 \end{cases}$$

O programa utilizado para resolver o sistema é mostrado na Figura 6. Ele utiliza o método da eliminação de Gauss e os resultados obtidos podem ser conferidos na Figura 7.

```
(8.0000, 35.0000, 55.0000, 63.0000, err = 0.0159,
(8.0000, 36.0000, 56.0000, 64.0000, err = 0.0156,
(9.0000, 37.0000, 57.0000, 65.0000, err = 0.0154,
(9.0000, 37.0000, 58.0000, 66.0000, err = 0.0152,
(9.0000, 38.0000, 59.0000, 67.0000, err = 0.0149,
(9.0000, 39.0000, 60.0000, 68.0000, err = 0.0147,
(9.0000, 39.0000, 60.0000, 68.0000, err = 0.0000,
```

Figura 6: Resultados obtidos

```
import numpy as np

x = np.array([0,0,0,0])
x_ant = np.array([0,0,0,0])
eps = 0.001

A = np.array([[-4,1,0,0],
              [50,-125,75,0],
              [0,75,-300,225],
              [0,0,-225,225]])

b = np.array([0,0,0,2000])

x1 = lambda x2, x3, x4: (b[0] - A[0,1]*x2 - A[0,2]*x3 - A[0,3]*x4)/A[0,0]
x2 = lambda x1, x3, x4: (b[1] - A[1,0]*x1 - A[1,2]*x3 - A[1,3]*x4)/A[1,1]
x3 = lambda x1, x2, x4: (b[2] - A[2,0]*x1 - A[2,1]*x2 - A[2,3]*x4)/A[2,2]
x4 = lambda x1, x2, x3: (b[3] - A[3,0]*x1 - A[3,1]*x2 - A[3,2]*x3)/A[3,3]

err = 10.

while err>eps:

    x[0] = x1(x[1],x[2],x[3])
    x[1] = x2(x[0],x[2],x[3])
    x[2] = x3(x[0],x[1],x[3])
    x[3] = x4(x[0],x[1],x[2])
    err = np.amax(np.absolute(x-x_ant))/np.amax(np.absolute(x))

    print("(%.4f,"%x[0],"%.4f,"%x[1],"%.4f,"%x[2],"%.4f,"%x[3],"err = %.4f,"%err)

    x_ant = np.copy(x)
```

Figura 7: Programa que utiliza o método da eliminação de Gauss

Questão 4

Pergunta: De acordo com o princípio de Arquimedes, a força de flutuação é igual ao peso do fluido deslocado pela parte submersa de um objeto. Para o tronco de cone mostrado abaixo, use o método da bisseção e mais algum método de sua escolha para determinar a altura h_1 da parte que está acima da água. Use os seguintes valores para seus cálculos:$r_1 = 0,5m$, $r_2 = 1m$, $h = 1m$, $\rho_f = densidadedotronco = 200kg/m^3$ e $\rho_w = densidadedagua = 1000kg/m^3$. O volume do trondo de cone é dado por:

$$V = \frac{\pi h}{3}(r_1^2 + r_2^2 + r_1 r_2)$$

.

Questão 5

Pergunta: Em Termodinâmica sob determinadas condições a relação entre o calor Q fornecido a um gás e sua variação de temperatura $T_f - T_i$ é dada por

$$Q = nR[A(T_f - T_i) + \frac{B}{2}((T_f^2 - T_i^2) + \frac{C}{3}(T_f^3 - T_i^3)]$$

Para o gás metano $R = 8,314J/mol.K$, $A = 1.702$, $B = 9.081 \times 10^{-3}K^{-1}$, $C = -2,164 \times 10^{-6}K^{-2}$. Em uma câmara tem-se $n = 2mol$ de metano a temperatura $T_i = 300K$. Qual será a temperatura final T_f se $20kJ$ de energia é absorvido pelo gás?

Resposta: Substituindo os valores e reorganizando os termos, chegamos à seguinte equação: $1.702T_f + 4.7405 * 10^{-3}T_f^2 - 7.2133 * 10^{-7}T_f^3 = 2135.36$, que pode ser escrita na forma do polinômio $P(T_f) =$

$-7.2133 * 10^{-7} T_f^3 + 4.7405 * 10^{-3} T_f^2 + 1.702 T_f - 2135.36 = 0$. Por questão de simplificação, iremos usar a partir de agora $T_f = x$. Utilizando o teorema da localização de raízes no círculo, concluímos que as raízes do polinômio se encontram no intervalo de $x = 1$ a $x = 450451$ e, com a regra dos sinais de Descartes, que há somente uma raiz positiva. Para encontrar tal raiz, foi utilizado o seguinte programa em Python, que utiliza o método de Newton com a solução inicial $x_0 = 1000$.

```
import numpy as np

P = lambda x: -7.2133*10**(-7)*x**3+4.7405*10**(-3)*x**2+1.702*x-2135.36
dP = lambda x: -21.64*10**(-7)*x**2+9.481*10**(-3)*x+1.702

err = 10
x = 1000
x_ant = 0

while err>= 0.00001:

    x = x - P(x)/dP(x)
    err = err = np.amax(np.absolute(x-x_ant))/np.amax(np.absolute(x))

    print('x = '+str(x)+', err = '+str(err))

    x_ant = x
```

Figura 8: Programa que utiliza o método de Newton

Os resultados obtidos foram:

```
x = 602.4160106441956, err = 1.0
x = 534.1308354669465, err = 0.12784353690711947
x = 531.4871731704444, err = 0.00497408485087605
x = 531.4830851219297, err = 7.6917753907588e-06
```

Figura 9: Resultados obtidos