EP2 - Cálculo Numérico

Nicholas Funari Voltani - Nº USP: 9359365

23 de Setembro de 2019

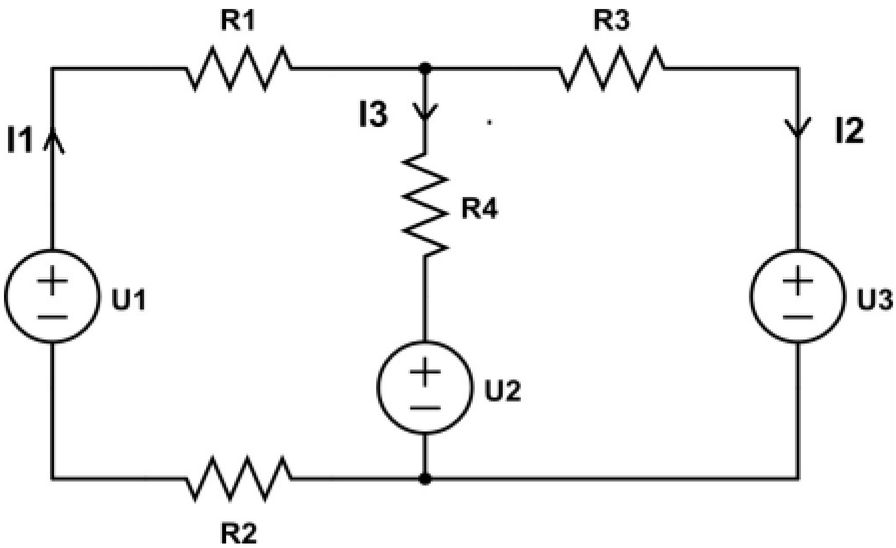

Figura 1: $R_1 = 7\,\Omega, R_2 = 4\,\Omega, R_3 = 5\,\Omega, R_4 = 1\,\Omega$;
$\mathrm{U}_1 = 20\,V, U_2 = 6\,V, U_3 = 1\,V$.

Temos que o sistema das correntes do circuito acima satisfazem

$$\begin{pmatrix} 0 & 5 & -1 \\ 11 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix} \begin{pmatrix} I_1 \\ I_2 \\ I_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 14 \\ 0 \end{pmatrix}$$

A demonstração, via leis de Kirchhoff, está no apêndice.

Quanto às resoluções/*prints* abaixo, o código de seus respectivos programas também estão no apêndice.

1 Resolução por Eliminação de Gauss (pivotamento parcial)

```
Metodo de Gauss (com pivotamento parcial)

Busca-se x tal que Ax = b, onde
A = [[ 0.  5. -1.]
 [11.  0.  1.]
 [ 1. -1. -1.]]

b = [ 5. 14.  0.]
--------------------------------------------------
Pivotamento parcial da 0-a coluna
A = [[11.  0.  1.]
 [ 0.  5. -1.]
 [ 1. -1. -1.]]

b=[14.  5.  0.]

0-a coluna escalonada
A=[[11.          0.          1.        ]
 [ 0.          5.         -1.        ]
 [ 0.         -1.         -1.09090909]]

b=[14.          5.         -1.27272727]

--------------------------------------------------
Pivotamento parcial da 1-a coluna
A = [[11.          0.          1.        ]
 [ 0.          5.         -1.        ]
 [ 0.         -1.         -1.09090909]]

b=[14.          5.         -1.27272727]

1-a coluna escalonada
A=[[11.          0.          1.        ]
 [ 0.          5.         -1.        ]
 [ 0.          0.         -1.29090909]]

b=[14.          5.         -0.27272727]

Obtencao da solucao x, por backsubstituting

x=[0.         0.         0.21126761]
x=[0.         1.         0.21126761]
x=[0.         1.04225352 0.21126761]
x=[1.27272727 1.04225352 0.21126761]
x=[1.25352113 1.04225352 0.21126761]
x=[1.25352113 1.04225352 0.21126761]

Resposta final: x = [1.25352113 1.04225352 0.21126761]
```

2 Resolução pelo método de Jacobi

```
Metodo de Jacobi
O sistema nao satisfaz o criterio das linhas

Chute inicial: x = (I_1, I_2, I_3) = [1 1 1]

Numero da iteracao: k = 1
 x_antes = [1 1 1]
 x_prox = [1.18181818 1.2        0.        ]
 erro = 0.19999999999999996

Numero da iteracao: k = 2
 x_antes = [1.18181818 1.2        0.        ]
 x_prox = [ 1.27272727  1.         -0.01818182]
 erro = 0.09090909090909083

Numero da iteracao: k = 3
 x_antes = [ 1.27272727  1.         -0.01818182]
 x_prox = [1.27438017 0.99636364 0.27272727]
 erro = 0.2909090909090908

Numero da iteracao: k = 4
 x_antes = [1.27438017 0.99636364 0.27272727]
 x_prox = [1.24793388 1.05454545 0.27801653]
 erro = 0.05818181818181811

Numero da iteracao: k = 5
 x_antes = [1.24793388 1.05454545 0.27801653]
 x_prox = [1.24745304 1.05560331 0.19338843]
 erro = 0.001057851239669505

Numero da iteracao: k = 6
 x_antes = [1.24745304 1.05560331 0.19338843]
 x_prox = [1.25514651 1.03867769 0.19184974]
 erro = 0.007693463561232017

Numero da iteracao: k = 7
 x_antes = [1.25514651 1.03867769 0.19184974]
 x_prox = [1.25528639 1.03836995 0.21646882]
 erro = 0.024619083395942765

Numero da iteracao: k = 8
 x_antes = [1.25528639 1.03836995 0.21646882]
 x_prox = [1.25304829 1.04329376 0.21691644]
 erro = 0.004923816679188508 5

Numero da iteracao: k = 9
 xi = [1.25304829 1.04329376 0.21691644]
 xf = [1.2530076  1.04338329 0.20975452]
 erro = 8.952393962169403e-05

Resposta final, com erro < 0.001: x = [1.2530076  1.04338329 0.20975452]
```

3 Resolução pelo método de Gauss-Seidel

```
Metodo de Gauss-Seidel

A = [[11  0  1]
 [ 0  5 -1]
 [ 1 -1 -1]]

b = [14  5  0]

O sistema satisfaz o criterio de Sassenfeld

Chute inicial: x = (I_1,I_2,I_3) = [1. 1. 1.]

Numero da iteracao: k = 1
 x_antes = [1. 1. 1.]
 x_prox = [ 1.18181818  1.2        -0.01818182]
 erro = 0.19999999999999996

Numero da iteracao: k = 2
 x_antes = [ 1.18181818  1.2        -0.01818182]
 x_prox = [1.27438017 0.99636364 0.27801653]
 erro = 0.29619834710743786

Numero da iteracao: k = 3
 x_antes = [1.27438017 0.99636364 0.27801653]
 x_prox = [1.24745304 1.05560331 0.19184974]
 erro = 0.05923966942148762

Numero da iteracao: k = 4
 x_antes = [1.24745304 1.05560331 0.19184974]
 x_prox = [1.25528639 1.03836995 0.21691644]
 erro = 0.02506670309405079

Numero da iteracao: k = 5
 x_antes = [1.25528639 1.03836995 0.21691644]
 x_prox = [1.2530076  1.04338329 0.20962431]
 erro = 0.0050133406188102025

Numero da iteracao: k = 6
 x_antes = [1.2530076  1.04338329 0.20962431]
 x_prox = [1.25367052 1.04192486 0.21174566]
 erro = 0.0021213474353976025

Numero da iteracao: k = 7
 x_antes = [1.25367052 1.04192486 0.21174566]
 x_prox = [1.25347767 1.04234913 0.21112854]
 erro = 0.0004242694870795205

Resposta final, com erro < 0.001: x = [1.25347767 1.04234913 0.21112854]
```

A Código do método de Gauss

Code 1: 1a) Código para o método de Gauss com pivotamento parcial.

```python
import numpy as np

A = np.array([
        [0.0,5,-1],
        [11,0,1],
        [1,-1,-1]])

b = np.array([5.0,14,0])

## Solucao buscada: x = A^(-1)*b

n=len(A) ## Pode ser alterado, depende de A_{nxn}
x = np.zeros([n]) ## Sera a solucao final

print("Metodo de Gauss (com pivotamento parcial)\n")
print("Busca-se x tal que Ax = b, onde")
print("A = {} \n\nb = {}".format(A,b))

for j in range(0,n-1): ## Ultima coluna (n-1) nao precisa ser analisada, 0<= j <= n-2
    i = j   ## Pivos sao elementos da diagonal

    ## Loop do pivotamento parcial
    for iprime in range(j,n):
        pivot_index = i
        if abs(A[iprime][j]) > abs(A[i][j]):
            pivot_index = iprime
        A[[i,pivot_index]] = A[[pivot_index,i]] #
        b[[i,pivot_index]] = b[[pivot_index,i]] ## Trocando linhas
    ##
    print("--------------------------------------------------")
    print("Pivotamento parcial da {}-a coluna".format(i))
    print("A = {} \n\nb={}\n".format(A,b))

    ## Escalonamento da coluna j
    for i_loop in range(i+1,n):  ## Lembrando que i == j
        b[i_loop] -= (A[i_loop][j]/A[i][j])*b[i]
        A[i_loop] -= (A[i_loop][j]/A[i][j])*A[i]
##
    print("{}-a coluna escalonada".format(j))
    print("A={}\n\nb={}\n".format(A,b))

## Logo, o sistema esta escalonado
print("Obtencao da solucao x, por backsubstituting\n")
for i in reversed(range(0,n)): ## Indo de baixo para cima na matriz
    x[i] = (1/A[i][i])*b[i]
    print("x={}".format(x))
    for j in reversed(range(i+1,n)):  ## Indo da frente para tras na linha i
        x[i] -= (A[i][j]*x[j])/A[i][i]
        print("x={}".format(x))
print("\nResposta final: x = {}".format(x))
```

B Código do método de Jacobi

Code 2: 1a) Código para o método de Jacobi.

```
import numpy as np

A = np.array([[0,5,-1],
      [11,0,1],
      [1,-1,-1]])

b = np.array([5,14,0])

A[[0,1]] = A[[1,0]]       ## Trocando linhas 0 e 1
b[0],b[1] = b[1],b[0]     ## Trocando elementos 0 e 1
n=3                       ## Pode ser alterado, depende de A_{nxn}
epsilon = 10**(-3)        ## Precisao desejada, pode ser alterada

print("Metodo de Jacobi")
######################################################################
## Verificar criterio das linhas
crit = True
for i in range(n):
    soma = 0
    for j in range(n):
        if j != i:
            soma += abs(A[i][j])
    if abs(A[i][i]) > abs(soma):
        crit = crit and True
    else:
        crit = crit and False

if crit:
    print("O sistema satisfaz o criterio das linhas")
else:
    print("O sistema nao satisfaz o criterio das linhas")
######################################################################

xi = np.array([1,1,1]) ## Chute inicial
k=1                    ## No da iteracao
xf = np.zeros([n])
for i in range(0,n):
    xf[i]=(b[i]/A[i][i])
    for j in range(0,n):
        if i != j:
            xf[i] -= A[i][j]*xi[j]/A[i][i]

erro = abs(max((xf-xi).min(), (xf-xi).max()))
print("\nChute inicial: x = (I_1, I_2, I_3) = {}\n".format(xi))
while erro > epsilon:
    print('Numero da iteracao: k = {}\n x_antes = {}\n x_prox = {}\n erro = {} \n'.format(k,
        xi,xf,erro))
    xi = xf
    xf=np.zeros([n])
    for i in range(0,n):
        xf[i]=(b[i]/A[i][i])
        for j in range(0,n):
            if i != j:
                xf[i] -= A[i][j]*xi[j]/A[i][i]
    erro = abs(max((xf-xi).min(), (xf-xi).max()))
    k+=1

print('Numero da iteracao: k = {}\n xi = {}\n xf = {}\n erro = {} \n'.format(k,xi,xf,erro))
    ## Ultimo da iteracao
print("Resposta final, com erro < {}: x = {}".format(epsilon, xf))
```

C Código do método de Gauss-Seidel

Code 3: 1a) Código para o método de Gauss-Seidel.

```
import numpy as np

A = np.array([[0,5,-1],
     [11,0,1],
     [1,-1,-1]])
b = np.array([5,14,0])

A[[0,1]] = A[[1,0]]    ## Trocando linhas 0 e 1
b[0],b[1] = b[1],b[0]  ## Trocando elementos 0 e 1

n=3                    ## Pode ser alterado, depende de A_{nxn}
epsilon = 10**(-3) ## Precisao desejada

print("Metodo de Gauss-Seidel")
print("\nA = {} \n\nb = {}\n".format(A,b))
##############################################################
## Verificar criterio de Sassenfeld
crit = True
beta = []
for i in range(0,n):
    soma = 0
    beta.append(soma)
    for j in range(0,n):
        if j<i :
            beta[i] += beta[j]*abs(A[i][j]/A[i][i])
        elif j>i:
            beta[i] += abs(A[i][j]/A[i][i])
    if beta[i] < 1:
        crit = crit and True
    else:
        crit = crit and False

if crit:
    print("O sistema satisfaz o criterio de Sassenfeld")
else:
    print("O sistema nao satisfaz o criterio de Sassenfeld")

##############################################################
k=1                              ## Numero da iteracao
xi = np.array([1.0,1.0,1.0]) ## Chute inicial
xf = xi.copy()

print("\nChute inicial: x = (I_1,I_2,I_3) = {}".format(xi))

## Criando x_1
for i in range(0,n):
    xf[i] = b[i]/A[i][i]
    for j in range(0,n):
        if j != i:
            xf[i] -= (A[i][j]*xf[j])/A[i][i]
##
erro = abs(max((xf-xi).min(), (xf-xi).max()))

## Iteracao de Gauss-Seidel
while erro > epsilon:
    print('\nNumero da iteracao: k = {}\n x_antes = {}\n x_prox = {}\n erro = {}'.format(k, xi,xf,erro))
    xi = xf.copy()
    for i in range(0,n):
        xf[i] = b[i]/A[i][i]
        for j in range(0,n):
            if j != i:
                xf[i] -= A[i][j]*xf[j]/A[i][i]
```

```
    erro = abs(max((xf-xi).min(), (xf-xi).max()))
    k += 1
##

## Ultimo valor da iteracao
print('\nNumero da iteracao: k = {}\n x_antes = {}\n x_prox = {}\n erro = {} \n'.format(k,xi,xf,erro))
print("Resposta final, com erro < {}: x = {}".format(epsilon, xf))
```