EP2 - Cálculo Numérico

Professor: Arnaldo Gammal
Monitor: Raphael

Aluno: Jhoão Gabriel Martins Campos de Almeida Arneiro
Nº USP: 10819721

Ao longo deste EP, nosso problema se consistirá em resolver o seguinte sistema:

$$\begin{bmatrix} 0 & 5 & -1 \\ 13 & 0 & 1 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 15 \\ 0 \end{bmatrix} \tag{1}$$

Item b

Para este item, vamos resolver o sistema (1) pelo método de Eliminação de Gauss usando pivotamento parcial. Nesta ocasião, optamos por transformar a primeira matriz em uma matriz aumentada na forma:

$$\left[\begin{array}{ccc|c} 0 & 5 & -1 & 4 \\ 13 & 0 & 1 & 15 \\ 1 & -1 & -1 & 0 \end{array}\right]$$

Abaixo está mostrado o processo de escalonamento desta matriz, para a obtenção do vetor solução $\boldsymbol{I}$ (para facilitar a visualização, vamos nos limitar aqui a apenas duas casas decimais):

$$\left[\begin{array}{ccc|c} 0 & 5 & -1 & 4 \\ 13 & 0 & 1 & 15 \\ 1 & -1 & -1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 13 & 0 & 1 & 15 \\ 0 & 5 & -1 & 4 \\ 1 & -1 & -1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 13 & 0 & 1 & 15 \\ 0 & 5 & -1 & 4 \\ 0 & -1 & -1.08 & -1.15 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{ccc|c} 13 & 0 & 1 & 15 \\ 0 & 5 & -1 & 4 \\ 0 & 0 & -1.28 & -0.35 \end{array}\right]$$

Tendo agora a matriz triangular superior, nós utilizamos a substituição para trás para obter como resultado:

$$\boldsymbol{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 1.132530 \\ 0.855422 \\ 0.277108 \end{bmatrix} \tag{2}$$

O programa utilizado para obter esta solução foi desenvolvido de modo que o usuário insira uma matriz $A_{n\times n}$ e um vetor resultante $\boldsymbol{b}$ n-dimensional, com isso ele calcula o vetor solução $\boldsymbol{x}$ da equação $A\boldsymbol{x} = \boldsymbol{b}$, e por fim imprime os elementos de $\boldsymbol{x}$. Este programa está listado abaixo:

```
def Escalona(A):
    ''' Recebe uma matriz aumentada A e por meio da Eliminacao por Gauss,
        utilizando pivotamento parcial, retorna a matriz triangular superior
        respectiva '''

    n = len(A) #Numero de linhas da matriz (igual ao tamanho da matriz
        quadrada
            #original)

    for k in range(n-1): #Varremos as n-1 primeiras colunas da matriz
                         #quadrada original para escalonar ela

        ######## Determinamos o indice do pivot l ########

        p = 0 #O pivot da k-esima coluna comeca igual a 0
        L = 0 #Armazenamos em qual linha esta o pivot

        #Para determinar o pivot, devemos checar o k-esimo elemento das linhas
        #k a n
        for l in range(k,n):

            #Se o elemento da l-esima linha da k-esima coluna for maior que p,
            #em modulo, ele se torna o novo p e guardamos em qual linha ele
            #se encontra
            if abs(A[l][k]) > p:

                p = A[l][k]
                L = l

        ##################################################

        ###### Trocamos a linhas k e L entre si ######

        #Copiamos quais sao os elementos de cada uma destas linhas
        Linha_k = A[k][:]
        Linha_l = A[L][:]

        #E entao trocamos eles entre si
        A[k] = Linha_l
        A[L] = Linha_k
```

```
        ##################################################

        #Tendo feito este processo inicial, comecamos a analisar linha
        #por linha (da (k+1)-esima linha a n-esima) e fazemos a operacao
        #Linha j = (Linha k)*(Multiplicador j) + Linha j para zerar a coluna
        #abaixo do pivot. Obs: Multiplicador j = -(elemento da j-esima
        #linha)/pivot

        for j in range(k+1,n):

            mult = -A[j][k]/p

            for i in range(k,n+1):

                A[j][i] = A[k][i]*mult + A[j][i]

    return A

def Backward(A):
    ''' Recebe uma matriz triangular superior aumentada e realiza uma
        substituicao para tras, de modo a nos retornar um vetor final,
        solucao do sistema '''

    n = len(A)
    Sol = []

    #Iniciamos com um vetor solucao de dimensao n e todos elementos iguais a 0
    for s in range(n):

        Sol.append(0)

    for i in range(n-1,-1,-1):

        #Iniciamos dividindo o valor que multiplica a variavel x_i
        #de modo a ficar com uma expressao do tipo:
        #x_i + a_(i+1)j x_(i+1)j + ... + a_nj x_j = b_i
        fator = 1/A[i][i]
        soma = 0

        #Fazemos a substituicao das variaveis ja conhecidas e a somamos entre
            si
        for j in range(i+1,n):

            soma = soma + Sol[j]*A[i][j]

        #Calculamos entao o valor de x_i que eh solucao de nosso problema
        Sol[i] = fator*(A[i][n] - soma)

    return Sol

#Iniciamos com uma matriz quadrada n x n A, um vetor resultante b e um vetor
```

```
#vazio x
A = [[0,5,-1],[13,0,1],[1,-1,-1]]
b = [4,15,0]
x = []

n = len(A)

#Transformamos nossa matriz A em uma matriz aumentada
for i in range(n):

    A[i].append(b[i])

#Por fim, escalonamos a matriz A (aumentada) e entao realizamos uma
    substituicao
#para tras para determinar o nosso vetor solucao x
A = Escalona(A)
x = Backward(A)

print("A solucao deste sistema eh dada por: I_1 = %f, I_2 = %f e I_3 = %f" %(x
    [0],x[1],x[2]))

>>> Pivotamento:
>>> [13, 0, 1, 15]
>>> [0, 5, -1, 4]
>>> [1, -1, -1, 0]
>>> Soma e multiplicacao:
>>> [13, 0, 1, 15]
>>> [0.0, 5.0, -1.0, 4.0]
>>> [1, -1, -1, 0]
>>> Soma e multiplicacao:
>>> [13, 0, 1, 15]
>>> [0.0, 5.0, -1.0, 4.0]
>>> [0.0, -1.0, -1.0769230769230769, -1.153846153846154]
>>> Pivotamento:
>>> [13, 0, 1, 15]
>>> [0.0, 5.0, -1.0, 4.0]
>>> [0.0, -1.0, -1.0769230769230769, -1.153846153846154]
>>> Soma e multiplicacao:
>>> [13, 0, 1, 15]
>>> [0.0, 5.0, -1.0, 4.0]
>>> [0.0, 0.0, -1.2769230769230768, -0.3538461538461539]
>>> A solucao deste sistema eh dada por: I_1 = 1.132530, I_2 = 0.855422 e I_3
    = 0.277108
```

Item c

Para este e o próximo item vamos permutar as duas primeiras linhas do sistema (1), e utilizaremos no lugar o sistema

$$\begin{bmatrix} 13 & 0 & 1 \\ 0 & 5 & -1 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 4 \\ 0 \end{bmatrix} \tag{3}$$

Queremos agora resolver este sistema pelo método de Jacobi, isto é, queremos resolver uma equação do tipo $A\boldsymbol{x} = \boldsymbol{b}$ (onde A é uma matriz $n \times n$, e ambos $\boldsymbol{x}$ e $\boldsymbol{b}$ são vetores n-dimensionais), a partir de um vetor inicial $\boldsymbol{x}^{(0)}$ arbitrário, através de um modo iterativo, que consiste em determinar o próximo vetor $\boldsymbol{x}^{(k)}$, com $k = 1, 2, \ldots$, a partir da relação de recursão

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right) \tag{4}$$

onde $x_i^{(k)}$ representa a i-ésima entrada do vetor $\boldsymbol{x}^{(k)}$, b_i é a i-ésima entrada do vetor $\boldsymbol{b}$ e a_{ij} é o elemento da i-ésima linha e j-ésima coluna da matriz A. Por se consistir de um método iterativo, devemos, antes de tudo, ver se este método é convergente para este sistema.

Para analisar a convergência, podemos fazer uso do Critério de Linhas, que é uma condição suficiente para nos garantir que o sistema irá convergir. Este critério nos diz que, se para alguma permutação das linhas do nosso sistema, ele obedecer que:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \quad \forall\, i = 1, \ldots, n \tag{5}$$

então ele irá convergir utilizando o método de Jacobi. Em outras palavras, os elementos diagonais de cada linha, devem ser maiores, em módulo, do que a soma dos módulos dos outros termos da mesma linha. Note que, se o Critério de Linhas não for satisfeito, o sistema ainda poderá convergir, contudo nós não teremos como saber isso apenas com esta informação. Como podemos ainda observar, vemos que os maiores termos de cada linha estão na diagonal do sistema (3), sendo assim, essa é a permutação com maiores chances de obedecer o Critério de Linhas. Vejamos então se isto ocorre:

$i = 1$

$$|a_{11}| = |13| = 13$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| = |a_{12}| + |a_{13}| = |0| + |1| = 1$$

$$\Rightarrow 13 > 1$$

$i = 2$

$$|a_{22}| = |5| = 5$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| = |a_{21}| + |a_{23}| = |0| + |-1| = 1$$

$$\Rightarrow 5 > 1$$

$i = 3$

$$|a_{33}| = |-1| = 1$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| = |a_{31}| + |a_{32}| = |1| + |-1| = 2$$

$$\Rightarrow 1 < 2$$

Como podemos ver, a relação (5) falha para $i = 3$, e portanto este sistema não satisfaz o Critério de Linhas. Poderíamos, se quiséssemos, realizar uma decomposição da matriz A, em uma combinação de uma matriz LU, multiplicada por uma matriz diagonal D e ver se no limite de $k \to \infty$, ela iria a 0, contudo, como veremos adiante, isso não será necessário, pois mesmo não satisfazendo o Critério de Linhas, o sistema ainda assim irá convergir. Vejamos então que, para um critério de parada dado por $\max |x_i^{(k+1)} - x_i^{(k)}| < \epsilon$, para $\epsilon = 10^{-4}$, o sistema irá convergir.

Com relação ao nosso problema em questão, nós iniciamos nossa iteração com um vetor $\boldsymbol{x_0} = (1, 1, 1)$ e, para acompanhar os valores do passo de iteração k, dos elementos I_1, I_2 e I_3 do vetor $\boldsymbol{I}$, e do erro $e_k = \max |x_i^{(k+1)} - x_i^{(k)}|$, ao longo de cada iteração, nós sintetizamos todos estes valores na tabela abaixo.

k	I_1	I_2	I_3	e_k
0	1.0000	1.0000	1.0000	
1	1.0769	1.0000	-0.0000	0.0769
2	1.1539	0.8000	0.0769	0.0769
3	1.1479	0.8154	0.3539	0.2769
4	1.1266	0.8708	0.3325	0.0554
5	1.1283	0.8665	0.2559	0.0016
6	1.1342	0.8512	0.2618	0.0059
7	1.1337	0.8524	0.2830	0.0212
8	1.1321	0.8566	0.2814	0.0043
9	1.1322	0.8563	0.2755	0.0001
10	1.1327	0.8551	0.2759	0.0005
11	1.1326	0.8552	0.2776	0.0016
12	1.1325	0.8555	0.2774	0.0003
13	1.1325	0.8555	0.2770	0.0000

Como podemos notar, o resultado converge e, dada a aproximação de 4 casas decimais, corresponde muito bem ao resultado obtido no item (b). O programa utilizado para implementar este método está listado abaixo.

```
#Constantes
epsilon = 1e-4

#Funcoes a serem usadas
def Jacobi(A,x_0,b):
    '''Recebe uma matriz A_nxn, um vetor inicial x_0 e um vetor resultante b
       e resolve o sistema Ax = b a partir metodo de Jacobi para um criterio
       de parada max|x_i(k+1) - x_i(k)| < epsilon, e imprime o numero da
           iteracao
       k, junto com os valores de x_i e o erro referente a cada iteracao. Ao
       fim, retorna o vetor solucao'''

    #Iniciamos com um passo k = 0 e um erro arbitrariamente grande por nao
    #possuirmos um vetor x_{n-1}
    k = 0
    erro = 10
    x_k = x_0[:]
    n = len(x_0)

    #Imprimimos os primeiros valores para k, I_1, I_2, I_3. Pelo erro ter sido
    #escolhido arbitrariamente, nao o incluimos na listagem
    print("k = %i, I_1 = %f, I_2 = %f, I_3 = %f, erro = --" %(k,x_k[0],x_k[1],
        x_k[2]))

    #Realizaremos iteracoes ate que o erro seja menor que a precisao definida
    while erro > epsilon:

        #Ao comeco da iteracao, colocamos o vetor x_{k-1} como igual ao vetor
        #da iteracao anterior
```

```
        x_k_1 = x_k[:]

        #Fazemos o calculo para cada componente i do vetor x
        for i in range(n):

            #Utilizamos uma variavel para guardar os termos referentes ao
            #somatorio
            soma = 0

            for j in range(n):

                #Excluimos apenas do somatorio o termo quando i = j
                if j == i:

                    continue

                #Fazemos a soma em cima dos valores de x_{k-1}, calculados na
                #iteracao anterior
                soma = soma + x_k_1[j]*A[i][j]

            #Calculamos o fator que multiplica os termos entre parenteses
            #e entao determinamos o novo valor de x_i
            fator = 1/A[i][i]
            x_k[i] = fator*(b[i] - soma)

        #Tendo determinado os novos valores de x para esta iteracao,
        #calculamos o erro referente a ela
        erro = 0

        for i in range(n):

            er = x_k[i] - x_k_1[i]

            if er > erro:

                erro = er

        #Atualizamos o numero do passo
        k = k + 1

        #Imprimimos os valores desta iteracao de k, do vetor I e do erro
        print("k = %i, I_1 = %f, I_2 = %f, I_3 = %f, erro = %f" %(k, x_k[0], x_k
            [1], x_k[2], erro))

    return(x_k)

#Inserimos os valores de A, b e do chute inicial x_0
A = [[13,0,1],[0,5,-1],[1,-1,-1]]
b = [15,4,0]
x_0 = [1,1,1]
```

```
#Determinamos o vetor solucao a partir do metodo de Jacobi
x = Jacobi(A, x_0, b)

print("A solucao final eh dada por I_1 = %f, I_2 = %f, I_3 = %f" %(x[0],x[1],x
    [2]))
```

```
>>> k = 0, I_1 = 1.000000, I_2 = 1.000000, I_3 = 1.000000, erro = --
>>> k = 1, I_1 = 1.076923, I_2 = 1.000000, I_3 = -0.000000, erro = 0.076923
>>> k = 2, I_1 = 1.153846, I_2 = 0.800000, I_3 = 0.076923, erro = 0.076923
>>> k = 3, I_1 = 1.147929, I_2 = 0.815385, I_3 = 0.353846, erro = 0.276923
>>> k = 4, I_1 = 1.126627, I_2 = 0.870769, I_3 = 0.332544, erro = 0.055385
>>> k = 5, I_1 = 1.128266, I_2 = 0.866509, I_3 = 0.255858, erro = 0.001639
>>> k = 6, I_1 = 1.134165, I_2 = 0.851172, I_3 = 0.261757, erro = 0.005899
>>> k = 7, I_1 = 1.133711, I_2 = 0.852351, I_3 = 0.282993, erro = 0.021236
>>> k = 8, I_1 = 1.132077, I_2 = 0.856599, I_3 = 0.281360, erro = 0.004247
>>> k = 9, I_1 = 1.132203, I_2 = 0.856272, I_3 = 0.275479, erro = 0.000126
>>> k = 10, I_1 = 1.132655, I_2 = 0.855096, I_3 = 0.275931, erro = 0.000452
>>> k = 11, I_1 = 1.132621, I_2 = 0.855186, I_3 = 0.277560, erro = 0.001629
>>> k = 12, I_1 = 1.132495, I_2 = 0.855512, I_3 = 0.277434, erro = 0.000326
>>> k = 13, I_1 = 1.132505, I_2 = 0.855487, I_3 = 0.276983, erro = 0.000010
>>> A solucao final eh dada por I_1 = 1.132505, I_2 = 0.855487, I_3 = 0.276983
```

Item d

Vamos agora resolver o sistema (3) a partir do método de Gauss-Seidel. Este método é muito similar ao método de Jacobi empregado no item anterior, contudo desta vez utilizaremos a seguinte relação de recursão

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \tag{6}$$

Isto é, ao invés de calcular o novo vetor $\boldsymbol{x}^{(k+1)}$ a partir do vetor $\boldsymbol{x}^{(k)}$, nós calculamos ele utilizando todos os valores $x_i^{(k+1)}$ que já tenham sido previamente calculados, junto com os valores restantes de $\boldsymbol{x}^{(k)}$. Em termos computacionais, isto significa apenas uma troca na expressão (4) do elemento $x_j^{(k)}$ pelo próprio elemento $x_j^{(k+1)}$.

Podemos novamente, discutir a convergência deste método para este sistema. Já vimos no item (c) que nossa matriz não satisfaz o Critério de Linhas, e nós apenas verificamos que o método de Jacobi convergiu, mas sem provar sua convergência. Mas agora que vamos implementar o método de Gauss-Seidel, nós podemos tentar aplicar um segundo critério de convergência, menos restritivo que o Critério de Linhas, que é o chamado Critério de Sassenfeld. O Critério de Sassenfeld nos diz que, definindo β_i's como

$$\begin{aligned}\beta_1 &\equiv \frac{|a_{12}| + |a_{13}| + \ldots + |a_{1n}|}{|a_{11}|} \\ \beta_i &\equiv \frac{\sum_{j=1}^{i-1} \beta_j |a_{ij}| + \sum_{j=i+1}^{n} |a_{ij}|}{|a_{ii}|}\end{aligned} \tag{7}$$

então o método de Gauss-Seidel irá convergir se $\beta_i < 1$, $\forall\, i = 1, \ldots, n$. Vejamos então se o sistema (3) obedece o critério de Sassenfeld:

$i = 1$

$$\beta_1 = \frac{|a_{12}| + |a_{13}|}{|a_{11}|} = \frac{|0| + |1|}{|13|} = \frac{|1|}{|13|} = \frac{1}{13} < 1$$

$i = 2$

$$\beta_2 = \frac{\beta_1 |a_{21}| + |a_{23}|}{|a_{22}|} = \frac{\frac{1}{13}|0| + |-1|}{|5|} = \frac{1}{5} < 1$$

$i = 3$

$$\beta_3 = \frac{\beta_1 |a_{31}| + \beta_2 |a_{32}|}{|a_{33}|} = \frac{\frac{1}{13}|1| + \frac{1}{5}|-1|}{|-1|} = \frac{1}{13} + \frac{1}{5} = \frac{18}{65} < 1$$

Podemos ver então que o sistema (3) satisfaz o Critério de Sassenfeld, e portanto nós podemos garantir que ele convergirá se utlizarmos o método de Gauss-Seidel.

Utilizaremos o mesmo critério de parada do item (b), dado por $\max |x_i^{(k+1)} - x_i^{(k)}| < \epsilon$, com $\epsilon = 10^{-4}$, o mesmo vetor inicial $\boldsymbol{x_0} = (1, 1, 1)$ e também imprimiremos os valores de k, I_1, I_2, I_3 e do erro e_k.

O código escrito para este método foi escrito em cima do código anterior, mas mudando, essencialmente, a linha 46, referente ao cálculo do somatório, desta vez utilizando os valores já calculados do vetor $\boldsymbol{x}^{(k)}$ dentro do mesmo laço, ao invés dos valores calculados no laço anterior. E é claro, este programa também é capaz de resolver qualquer equação na forma $A\boldsymbol{x} = \boldsymbol{b}$, com A sendo uma matriz $n \times n$ e $\boldsymbol{x}$ e $\boldsymbol{b}$ vetores n-dimensionais.

A tabela com os valores calculados e o código utilizado para obtê-los está exposto abaixo.

k	I_1	I_2	I_3	e_k
0	1.0000	1.0000	1.0000	
1	1.0769	1.0000	0.0769	0.0769
2	1.1479	0.8154	0.3325	0.2556
3	1.1283	0.8665	0.2618	0.0511
4	1.1337	0.8524	0.2814	0.0196
5	1.1322	0.8563	0.2759	0.0039
6	1.1326	0.8552	0.2774	0.0015
7	1.1325	0.8555	0.2770	0.0003
8	1.1325	0.8554	0.2771	0.0001
9	1.1325	0.8554	0.2771	0.0000

```
#Constantes
epsilon = 1e-4

#Funcoes a serem usadas
def Gauss_Seidel(A,x_0,b):
    '''Recebe uma matriz A_nxn, um vetor inicial x_0 e um vetor resultante b
       e resolve o sistema Ax = b a partir metodo de Gauss-Seidel para um
       criterio de parada max|x_i(k+1) - x_i(k)| < epsilon, e imprime o
       numero da iteracao k, junto com os valores de x_i e o erro referente a
       cada iteracao. Ao fim, retorna o vetor solucao'''

    #Iniciamos com um passo k = 0 e um erro arbitrariamente grande por nao
    #possuirmos um vetor x_{n-1}
    k = 0
    erro = 10
    x_k = x_0[:]
    n = len(x_0)

    #Imprimimos os primeiros valores para k, I_1, I_2, I_3. Pelo erro ter sido
    #escolhido arbitrariamente, nao o incluimos na listagem
    print("k = %i, I_1 = %f, I_2 = %f, I_3 = %f, erro = --" %(k,x_k[0],x_k[1],
        x_k[2]))

    #Realizaremos iteracoes ate que o erro seja menor que a precisao definida
    while erro > epsilon:

        #Ao comeco da iteracao, colocamos o vetor x_{k-1} como igual ao vetor
        #da iteracao anterior - usado apenas para o calculo do erro
        x_k_1 = x_k[:]

        #Fazemos o calculo para cada componente i do vetor x
        for i in range(n):

            #Utilizamos uma variavel para guardar os termos referentes ao
            #somatorio
            soma = 0
```

```
            for j in range(n):

                #Excluimos apenas do somatorio o termo quando i = j
                if j == i:

                    continue

                #Fazemos a soma em cima dos valores do proprio x_k, calculados
                #ao longo deste loop
                soma = soma + x_k[j]*A[i][j]

            #Calculamos o fator que multiplica os termos entre parenteses
            #e entao determinamos o novo valor de x_i
            fator = 1/A[i][i]
            x_k[i] = fator*(b[i] - soma)

        #Tendo determinado os novos valores de x para esta iteracao,
        #calculamos o erro referente a ela
        erro = 0

        for i in range(n):

            er = x_k[i] - x_k_1[i]

            if er > erro:

                erro = er

        #Atualizamos o numero do passo
        k = k + 1

        #Imprimimos os valores desta iteracao de k, do vetor I e do erro
        print("k = %i, I_1 = %f, I_2 = %f, I_3 = %f, erro = %f" %(k,x_k[0],x_k
            [1],x_k[2],erro))

    return(x_k)

#Inserimos os valores de A, b e do chute inicial x_0
A = [[13,0,1],[0,5,-1],[1,-1,-1]]
b = [15,4,0]
x_0 = [1,1,1]

#Determinamos o vetor solucao a partir do metodo de Gauss-Seidel
x = Gauss_Seidel(A, x_0, b)

print("A solucao final eh dada por I_1 = %f, I_2 = %f, I_3 = %f" %(x[0],x[1],x
    [2]))
```

```
>>> k = 0, I_1 = 1.000000, I_2 = 1.000000, I_3 = 1.000000, erro = --
>>> k = 1, I_1 = 1.076923, I_2 = 1.000000, I_3 = 0.076923, erro = 0.076923
>>> k = 2, I_1 = 1.147929, I_2 = 0.815385, I_3 = 0.332544, erro = 0.255621
```

```
>>> k = 3, I_1 = 1.128266, I_2 = 0.866509, I_3 = 0.261757, erro = 0.051124
>>> k = 4, I_1 = 1.133711, I_2 = 0.852351, I_3 = 0.281360, erro = 0.019603
>>> k = 5, I_1 = 1.132203, I_2 = 0.856272, I_3 = 0.275931, erro = 0.003921
>>> k = 6, I_1 = 1.132621, I_2 = 0.855186, I_3 = 0.277434, erro = 0.001503
>>> k = 7, I_1 = 1.132505, I_2 = 0.855487, I_3 = 0.277018, erro = 0.000301
>>> k = 8, I_1 = 1.132537, I_2 = 0.855404, I_3 = 0.277133, erro = 0.000115
>>> k = 9, I_1 = 1.132528, I_2 = 0.855427, I_3 = 0.277102, erro = 0.000023
>>> A solucao final eh dada por I_1 = 1.132528, I_2 = 0.855427, I_3 = 0.277102
```