Jardel_Metodos_Numericos_Unidade_2_Semana_1

October 21, 2020

**

Disciplina:** Métodos Numéricos

**

Semestre:** 2020.2

**

Aluno:** Jardel Brandon de Araujo Regis

**

Mátricula:** 201621250014

**

2ª Unidade:** Álgebra Linear Computacional

[1]:
```python
import numpy as np
```

0.0.1 Introdução à sistemas de equações lineares

$$\begin{aligned} 3x_1 + 4x_2 - 5x_3 + x_4 &= -10 \\ x_2 + x_3 - 2x_4 &= -1 \\ 4x_3 - 5x_4 &= 3 \\ 2x_4 &= 2 \end{aligned}$$

Exercício 1. Verifique, refazendo o exemplo acima, se a equação da substituição retroativa, descrita na equação 20, de fato funciona.

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij} x_j}{a_{ii}}, \quad \forall i = m, \ldots, 1$$

20.

Resposta:

x_n, pode ser encontrada diretamente, simplesmente fazendo

$$a_{nn} x_n = b_n \Rightarrow x_n = \frac{b_n}{a_{nn}} \tag{1}$$

$$\begin{aligned} x_n &= \tfrac{2}{2} \\ x_4 &= 1 \end{aligned}$$

Para a penúltima equação , $a_{(n-1)(n-1)}x_{n-1} + a_{(n-1)n}x_n = b_{n-1}$, o valor de uma incógnita já é conhecido, x_n. Dessa forma, para encontrar a incógnita que falta, basta isolar a incógnita cujo valor ainda não se conhece

$$a_{(n-1)(n-1)}x_{n-1} + a_{(n-1)n}x_n = b_{n-1} \Rightarrow x_{n-1} = \frac{b_{n-1} - a_{(n-1)n}x_n}{a_{(n-1)(n-1)}} \tag{2}$$

substituindo o valor de x_n encontrado no passo anterior.

$$\begin{aligned} x_{n-1} &= \tfrac{(3-(-5)*1)}{4} \\ x_{n-1} &= \tfrac{3+5}{4} \\ x_{n-1} &= \tfrac{8}{4} \\ x_3 &= 2 \end{aligned}$$

$$\begin{aligned} x_{n-2} &= \tfrac{(-1-((1*2)+(-2*1))}{1} \\ x_{n-2} &= \tfrac{-1}{1} \\ x_2 &= -1 \end{aligned} \tag{3}$$

$$\begin{aligned} x_{n-3} &= \tfrac{-10-((4*-1)+(-5*2)+(1*1))}{3} \\ x_{x-3} &= \tfrac{-10-(-4-10+1}{3} \\ x_{n-3} &= \tfrac{-10-(-13)}{3} \\ x_{n-3} &= \tfrac{-10+13)}{3} \\ x_{n-3} &= \tfrac{3)}{3} \\ x_1 &= 1 \end{aligned} \tag{4}$$

Logo,

$$\begin{aligned} x_1 &= 1 \\ x_2 &= -1 \\ x_3 &= 2 \\ x_4 &= 1 \end{aligned} \tag{5}$$

Exercício 2. Faça um programa em Python que calcule a solução de um sistema triangular superior usando substituição retroativa. Para representação das matrizes e vetores use exclusivamente Numpy.

[2]:
```python
def resolver_sistemas_lineares_triangulares(A, b, inferior=False):
    n = len(b)
    x = np.empty(n)
    if(inferior == True):
        x[0] = b[0] / A[0][0]
        for i in range(1, n):
            x[i] = (b[i] - np.sum(A[i][:i] * x[:i])) / A[i, i]
    else:
        x[-1] = b[-1] / A[-1][-1]
        for i in range(n - 2, -1, -1):
            x[i] = (b[i] - np.sum(A[i][i + 1:] * x[i + 1:])) / A[i][i]
    return x
```

[3]:
```python
A = np.array([[3, 4, -5, 1], [0, 1, 1, -2], [0, 0, 4, -5], [0, 0, 0, 2]])
b = np.array([-10, -1, 3, 2])
```

[4]:
```python
x = resolver_sistemas_lineares_triangulares(A, b)
x
```

[4]:
```
array([ 1., -1.,  2.,  1.])
```

[5]:
```python
A.dot(x)
```

[5]:
```
array([-10.,  -1.,   3.,   2.])
```

Exercício 3. Faça um programa em Python que calcule a solução de um sistema triangular inferior usando substituição progressiva. Para representação das matrizes e vetores use exclusivamente Numpy.

$$
\begin{aligned}
3x_1 &= 4 \\
2x_1 + x_2 &= 2 \\
x_1 + x_3 &= 4 \\
x_1 + x_2 + x_3 + x_4 &= 2
\end{aligned}
$$

[6]:
```python
A = np.array([[3,0,0,0],[2,1,0,0],[1,0,1,0],[1,1,1,1]])
b = np.array([4,2,4,2])
```

[7]:
```python
x = resolver_sistemas_lineares_triangulares(A, b, True)
x
```

[7]:
```
array([ 1.33333333, -0.66666667,  2.66666667, -1.33333333])
```

[8]:
```python
A.dot(x)
```

[8]:
```
array([4., 2., 4., 2.])
```

Exercício 4. Compare os resultados das implementações de suas funções para a solução de sistemas triangulares com a função do scipy. Os resultados são diferentes? Experimente com outros exemplos.

[9]:
```
import scipy.linalg as sla
import numpy as np
```

$$\begin{aligned} 3x_1 + 4x_2 - 5x_3 + x_4 &= -10 \\ x_2 + x_3 - 2x_4 &= -1 \\ 4x_3 - 5x_4 &= 3 \\ 2x_4 &= 2 \end{aligned}$$

[10]:
```
A = np.array([[3, 4, -5, 1],[0, 1, 1, -2],[0, 0, 4, -5],[0, 0, 0, 2]])
b = np.array([-10, -1, 3, 2])
```

[11]:
```
x = sla.solve_triangular(A,b)
x
```

[11]:
```
array([ 1., -1.,  2.,  1.])
```

[12]:
```
y = resolver_sistemas_lineares_triangulares(A, b)
y
```

[12]:
```
array([ 1., -1.,  2.,  1.])
```

[13]:
```
print(np.array_equal(x, y))
```

```
True
```

$$\begin{aligned} 3x_1 &= 4 \\ 2x_1 + x_2 &= 2 \\ x_1 + x_3 &= 4 \\ x_1 + x_2 + x_3 + x_4 &= 2 \end{aligned}$$

[14]:
```
A = np.array([[3,0,0,0],[2,1,0,0],[1,0,1,0],[1,1,1,1]])
b = np.array([4,2,4,2])
```

[15]:
```
x = sla.solve_triangular(A,b, lower=True)
x
```

[15]:
```
array([ 1.33333333, -0.66666667,  2.66666667, -1.33333333])
```

[16]:
```
y = resolver_sistemas_lineares_triangulares(A, b, True)
y
```

[16]: array([1.33333333, -0.66666667, 2.66666667, -1.33333333])

[17]:
```python
print(np.array_equal(x, y))
```

False

Outro Exemplo:

$$\begin{cases} x_1 + 2x_2 + x_3 = 20 \\ \qquad x_2 + 2x_3 = 11 \\ \qquad\qquad x_3 = 3 \end{cases}$$

[18]:
```python
A = np.array([[1, 2, 1], [0, 1, 2], [0, 0, 1]])
b = np.array([20, 11, 3])
```

[19]:
```python
x = sla.solve_triangular(A,b)
x
```

[19]: array([7., 5., 3.])

[20]:
```python
y = resolver_sistemas_lineares_triangulares(A, b)
y
```

[20]: array([7., 5., 3.])

[21]:
```python
print(np.array_equal(x, y))
```

True

Outro Exemplo:

$$\begin{pmatrix} 2 & 1 & 3 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 9 \\ 1 \\ 2 \end{pmatrix}$$

[22]:
```python
A = np.array([[2, 1, 3], [0, -1, 1], [0, 0, 1]])
b = np.array([9, 1, 2])
```

[23]:
```python
x = sla.solve_triangular(A,b)
x
```

[23]: array([1., 1., 2.])

[24]:
```python
y = resolver_sistemas_lineares_triangulares(A, b)
y
```

[24]: array([1., 1., 2.])

[25]:
```python
print(np.array_equal(x, y))
```

```
True
```

0.0.2 Introdução à sistemas de equações lineares - Complemento

Exercício 1.1 Classifique os sistemas abaixo com relação a quantidade e existência de soluções.

a)

$$\begin{aligned} x + 2y + 3z &= 1 \\ 4x + 5y + 6z &= 1 \\ 7x + 8y + 9z &= 1 \end{aligned}$$

resolução a)

[26]:
```python
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9,]])
b = np.array([1, 1, 1])
```

[27]:
```python
np.linalg.matrix_rank(A)
```

[27]:
```
2
```

[28]:
```python
np.linalg.matrix_rank(np.c_[A, b])
```

[28]:
```
2
```

[29]:
```python
A.shape[1]
```

[29]:
```
3
```

Como o posto da matriz dos coeficientes é igual ao posto da matriz ampliada que são menores que o número de incógnitas: $P_C = P_A < número de incógnitas. Logo o sistema possuí infinitas soluções (Possível e Inderteminados)

b)

$$2x + 3y = 10$$

$$-4x - 6y = -10$$

resolução b)

[30]:
```python
A = np.array([[2, 3], [-4, -6]])
b = np.array([10, -10])
```

[31]:
```python
np.linalg.matrix_rank(A)
```

[31]:
```
1
```

[32]:
```python
np.linalg.matrix_rank(np.c_[A, b])
```

[32]:
```
2
```

[33]:
```
A.shape[1]
```

[33]: 2

Como o posto da matriz dos coeficientes é menor que o posto da matriz ampliada: $P_C < P_A$. Logo o sistema não tem solução (Impossível)

Exemplo 3.2

$$\begin{aligned} 3x_1 + 4x_2 - 5x_3 + x_4 &= -10 \\ x_2 + x_3 - 2x_4 &= -1 \\ 4x_3 - 5x_4 &= 3 \\ 2x_4 &= 2 \end{aligned}$$

Exercício 1.2 Repita o exemplo 3.2 acima, porém fazendo cada passo da execução de forma explícita, comparando como o algoritmo 1.3 e a função sist_lin_tri_sup funcionam. Sugestão: faça isso de forma manuscrita. Ajudará a entender melhor cada passo.

Algoritmo 1.3 Substituição retroativa para sistemas triangulares superiores Entrada: matriz triangular superior de coeficientes A, vetor de termos independentes b Passo 1: criar um vetor com todos os valores iguais a zero: $x = 0$ Passo 2: calcular a incógnita de maior índice $x_n = \frac{b_n}{a_{nn}}$ para i variando de $n-1$ à 1 faça Passo 3: calcular os valores restantes do vetor solução

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}$$

Passo 4: atualizar os valores do vetor x

Resposta:

x_n, pode ser encontrada diretamente, simplesmente fazendo

$$a_{nn}x_n = b_n \Rightarrow x_n = \frac{b_n}{a_{nn}} \tag{6}$$

$$\begin{aligned} x_n = &\quad \tfrac{2}{2} \\ x_4 = &\quad 1 \end{aligned}$$

Para a penúltima equação ,$a_{(n-1)(n-1)}x_{n-1} + a_{(n-1)n}x_n = b_{n-1}$, o valor de uma incógnita já é conhecido, x_n. Dessa forma, para encontrar a incógnita que falta, basta isolar a incógnita cujo valor ainda não se conhece

$$a_{(n-1)(n-1)}x_{n-1} + a_{(n-1)n}x_n = b_{n-1} \Rightarrow x_{n-1} = \frac{b_{n-1} - a_{(n-1)n}x_n}{a_{(n-1)(n-1)}} \tag{7}$$

substituindo o valor de x_n encontrado no passo anterior.

$$\begin{aligned} x_{n-1} = &\quad \tfrac{(3-(-5)*1)}{4} \\ x_{n-1} = &\quad \tfrac{3+5}{4} \\ x_{n-1} = &\quad \tfrac{8}{4} \\ x_3 = &\quad 2 \end{aligned}$$

$$
\begin{aligned}
x_{n-2} = & \quad \frac{(-1-((1*2)+(-2*1))}{1} \\
x_{n-2} = & \quad \frac{-1}{1} \\
x_2 = & \quad -1
\end{aligned} \tag{8}
$$

$$
\begin{aligned}
x_{n-3} = & \quad \frac{-10-((4*-1)+(-5*2)+(1*1))}{3} \\
x_{x-3} = & \quad \frac{-10-(-4-10+1}{3} \\
x_{n-3} = & \quad \frac{-10-(-13)}{3} \\
x_{n-3} = & \quad \frac{-10+13)}{3} \\
x_{n-3} = & \quad \frac{3)}{3} \\
x_1 = & \quad 1
\end{aligned} \tag{9}
$$

Logo,

$$
\begin{aligned}
x_1 = & \quad 1 \\
x_2 = & \quad -1 \\
x_3 = & \quad 2 \\
x_4 = & \quad 1
\end{aligned} \tag{10}
$$

[34]:
```
def sist_lin_tri_sup(A,b):
    n = len(b)
    x = np.empty(n)
    x[-1] = b[-1]/A[-1, -1]
    for i in range(n-2, -1, -1):
        x[i] = (b[i] - np.sum(A[i,i+1:]*x[i+1:]))/A[i,i]
    return x
```

[35]:
```
A = np.array([[3, 4, -5, 1],[0, 1, 1, -2],[0, 0, 4, -5],[0, 0, 0, 2]])
b = np.array([-10, -1, 3, 2])
```

[36]:
```
x = sist_lin_tri_sup(A, b)
x
```

[36]:
```
array([ 1., -1.,  2.,  1.])
```