Jardel_Metodos_Numericos_Unidade_2_Semana_2

November 2, 2020

**

Disciplina:** Métodos Numéricos

**

Semestre:** 2020.2

**

Aluno:** Jardel Brandon de Araujo Regis

**

Mátricula:** 201621250014

**

2ª Unidade:** Álgebra Linear Computacional

[1]:
```
import numpy as np
import scipy.linalg as sla
```

0.0.1 Eliminação Gaussiana

[2]:
```
def elimin_gaussiana(a,b,imp = False, norma='inf'):
    import numpy as np #colocadas aqui para evitar problemas de namespace
    import scipy.linalg as sla #colocadas aqui para evitar problemas de␣
 ↪namespace

    #matriz estendida Me
    b = np.mat(b).T
    Me = np.concatenate((a,b), axis=1)

    if imp: print('Matriz estendida: \n',Me,'\n\n')
    if imp: print('='*80)
    for j in range(Me.shape[-1] - 2): #coluna
        if imp: print('Operação na coluna', j)
        if imp: print('='*80, '\n\n')
        for i in range(j+1, Me.shape[0]): #linha
            Me[i] = Me[i] - (Me[i,j]/Me[j,j])*Me[j] #operações com cada linha
            if imp: print('Operação na linha', i, ':',Me[i], '\n')
```

```
            if imp: print('Matriz escalonando: \n',Me,'\n')
        if imp: print('='*80)
    if imp: print('\nMatriz escalonada: \n',Me,'\n')

    Matriz_A = np.delete(Me, -1, axis=1) #matriz de coeficientes escalonada
    Vetor_b = Me[:,-1] #matriz de termos independentes escalonada
    Vetor_x = sla.solve_triangular(Matriz_A, Vetor_b) # solução do sistema␣
↪escalonado

    if imp: print('='*80,'\n\nSolução do sistema: \n', Vetor_x)

    #verificação do erro do sistema
    if norma is 'inf': norma = np.inf
    if norma is '-inf': norma = -np.inf
    Erro = sla.norm(b - a.dot(Vetor_x),ord = norma)

    return (Matriz_A, Vetor_b, Vetor_x, Erro)
```

[3]:
```
def elimin_gaussiana_pivoteamento_parcial(a, b, imp=False, norma='inf'):
    # matriz estendida Me
    b = np.mat(b).T
    Me = np.concatenate((a, b), axis=1)
    linhas, colunas = Me.shape
    Me_copia = np.copy(Me.astype('float'))
    a_copia = np.copy(a.astype('float'))
    b_copia = np.copy(b.astype('float'))

    if imp: print('Matriz estendida: \n', Me_copia, '\n\n')
    if imp: print('=' * 80)
    for j in range(colunas - 2):  # coluna
        maior_coluna_indice = np.argmax(np.abs(Me_copia[j:, j])) + j
        if maior_coluna_indice >= j:
            Me_copia[[j, maior_coluna_indice]] = Me_copia[[maior_coluna_indice,␣
↪j]]
            a_copia[[j, maior_coluna_indice]] = a_copia[[maior_coluna_indice,␣
↪j]]
            b_copia[[j, maior_coluna_indice]] = b_copia[[maior_coluna_indice,␣
↪j]]
            if imp: print("Trocando linha: ",maior_coluna_indice, "->",␣
↪Me_copia[maior_coluna_indice], "Pela linha: ", j, "->", Me_copia[j])
        if imp: print('Operação na coluna', j)
        if imp: print('=' * 80, '\n\n')
        for i in range(j + 1, linhas):  # linha
            Me_copia[i] = np.subtract(Me_copia[i], (Me_copia[i, j] /␣
↪Me_copia[j, j]) * Me_copia[j])  # operações com cada linha
            if imp: print('Operação na linha', i, ':', Me_copia[i], '\n')
```

```
            if imp: print('Matriz escalonando: \n', Me_copia, '\n')
        if imp: print('=' * 80)
    if imp: print('\nMatriz escalonada: \n', Me_copia, '\n')

    Matriz_A = np.delete(Me_copia, -1, axis=1)  # matriz de coeficientes␣
↪escalonada
    Vetor_b = Me_copia[:, -1]  # matriz de termos independentes escalonada
    Vetor_x = sla.solve_triangular(Matriz_A, Vetor_b)  # solução do sistema␣
↪escalonado

    if imp: print('=' * 80, '\n\nSolução do sistema: \n', Vetor_x)

    # verificação do erro do sistema
    if norma is 'inf': norma = np.inf
    if norma is '-inf': norma = -np.inf
    Erro = sla.norm(b_copia.transpose() - a_copia.dot(Vetor_x), ord=norma)

    return (Matriz_A, Vetor_b, Vetor_x, Erro)
```

[4]:
```
a = np.array([[2,3,-1],[4,4,-3],[2,-3,1]])
b = np.array([5,3,-1])
M = elimin_gaussiana(a,b)
M
```

[4]:
```
(matrix([[ 2,  3, -1],
         [ 0, -2, -1],
         [ 0,  0,  5]]),
 matrix([[ 5],
         [-7],
         [15]]),
 array([[1.],
        [2.],
        [3.]]),
 0.0)
```

[5]:
```
sla.solve(a,b)
```

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

Exercise 1 Interprete o erro apresentado acima e altere a função elimin_gaussiana, para que esse erro não ocorra. Essa abordagem de escolha do pivô como o elemento da coluna onde se realizará as operações que está na diagonal principal da matriz de coeficientes deixa o processo bastante sensível a propagação de erros durante as operações.

[6]:
```
a = np.array([[1,1,1],[4,4,2],[2,1,-1]])
b = np.array([1,2,0])
```

[7]:
```
elimin_gaussiana_pivoteamento_parcial(a,b)
```

[7]:
```
(array([[ 4. ,  4. ,  2. ],
        [ 0. , -1. , -2. ],
        [ 0. ,  0. ,  0.5]]),
 array([ 2. , -1. ,  0.5]),
 array([ 1., -1.,  1.]),
 0.0)
```

[8]:
```
sla.solve(a,b)
```

[8]:
```
array([ 1., -1.,  1.])
```

Example 3

Resolva o sistema:

$$\begin{aligned} x_1 + 4x_2 + 52x_3 &= 57 \\ 27x_1 + 110x_2 - 3x_3 &= 134 \\ 22x_1 + 2x_2 + 14x_3 &= 38 \end{aligned}$$

pelo método de eliminação gaussiana. Após escalonamento e aplicando, substituição reatroativa ao sistema escalonado, obtemos

$$x_1 = 4.5 \quad x_2 = 0 \quad x_3 = 1.01$$

bastante diferente de

$$x_1 = 1 \quad x_2 = 1 \quad x_3 = 1$$

a solução exata do sistema.

Exercise 2 Quantifique o erro da solução encontrada acima. Esse exemplo ilustra bem os efeitos de propagação de erros quando escalonamos matrizes usando o método de eliminação gaussiana convencional. Esse efeito pode ser mensurado melhor quando consideramos o problema com elementos com grande diferença de escala, como a seguir. Existem duas alternativas à essa escolha de pivô por posição:

pivotamento parcial: o pivô escolhido é o maior número na coluna a ser avaliada;

pivotamento total: o pivô escolhido é o maior número na matriz de coeficientes (ver livro do Chapra).

[9]:
```
a = np.array([[1, 4, 52], [27, 110, -3], [22, 2, 14]])

erro = sla.norm(a, ord=np.inf)
erro
```

[9]:
```
140.0
```

Exercise 3 Refaça o exemplo 3 usando eliminação gaussiana com pivotamento parcial e compare os erros do resultado obtido aqui com os do exemplo.

[10]:
```python
a = np.array([[1,4,52],[27,110,-3],[22,2,14]])
b = np.array([57,134,38])
```

[11]:
```python
elimin_gaussiana(a, b)
```

[11]:
```
(matrix([[     1,      4,     52],
         [     0,      2,  -1407],
         [     0,      0, -61631]]),
 matrix([[    57],
         [ -1405],
         [-61631]]),
 array([[1.],
        [1.],
        [1.]]),
 0.0)
```

[12]:
```python
elimin_gaussiana_pivoteamento_parcial(a,b)
```

[12]:
```
(array([[ 27.        , 110.        ,  -3.        ],
        [  0.        , -87.62962963,  16.44444444],
        [  0.        ,   0.        ,  52.09721048]]),
 array([134.        , -71.18518519,  52.09721048]),
 array([1., 1., 1.]),
 0.0)
```

[13]:
```python
sla.solve(a,b)
```

[13]:
```
array([1., 1., 1.])
```

O resultado obtido está de acordo com o cálculo análitico

Exercise 4 Implemente uma versão da função elimin_gaussiana que faça o pivotamento parcial. Compare os resultados e erros com os exemplos anteriores.

Example 4

Resolva o sistema

$$0.02x_1 + 0.01x_2 = 0.02$$
$$x_1 + 2x_2 + x_3 = 1$$
$$x_2 + 2x_3 + x_4 = 4$$
$$100x_3 + 200x_4 = 800$$

[14]:
```python
a = np.array([[0.02, 0.01, 0, 0], [1, 2, 1, 0], [0, 2, 0, 1], [0, 0, 100, 200]])
b = np.array([0.02, 1, 4, 800])
```

[15]:
```python
elimin_gaussiana(a, b)
```

[15]: (matrix([[2.00000000e-02, 1.00000000e-02, 0.00000000e+00,
          0.00000000e+00],
        [0.00000000e+00, 1.50000000e+00, 1.00000000e+00,
          0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, -1.33333333e+00,
          1.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
          2.75000000e+02]]),
 matrix([[2.0e-02],
         [0.0e+00],
         [4.0e+00],
         [1.1e+03]]),
 array([[1.],
        [0.],
        [-0.],
        [4.]]),
 0.0)

```
[16]: elimin_gaussiana_pivoteamento_parcial(a,b)
```

[16]: (array([[1.0e+00, 2.0e+00, 1.0e+00, 0.0e+00],
        [0.0e+00, 2.0e+00, 0.0e+00, 1.0e+00],
        [0.0e+00, 0.0e+00, 1.0e+02, 2.0e+02],
        [0.0e+00, 0.0e+00, 0.0e+00, 5.5e-02]]),
 array([1.0e+00, 4.0e+00, 8.0e+02, 2.2e-01]),
 array([1., 0., 0., 4.]),
 0.0)

```
[17]: sla.solve(a,b)
```

[17]: array([1., 0., 0., 4.])

O resultado obtido está de acordo com o cálculo análitico

Example 5

Resolva o sistema

$$\begin{aligned} x + y + z &= 1 \\ 2x + y - z &= 0 \\ 2x + 2y + z &= 1 \end{aligned}$$

```
[18]: a = np.array([[1, 1, 1], [ 2, 1, -1], [2, 2, 1]])
      b = np.array([1, 0, 1])
```

```
[19]: elimin_gaussiana(a, b)
```

[19]:
```
(matrix([[ 1,  1,  1],
         [ 0, -1, -3],
         [ 0,  0, -1]]),
 matrix([[ 1],
         [-2],
         [-1]]),
 array([[ 1.],
        [-1.],
        [ 1.]]),
 0.0)
```

[20]:
```
elimin_gaussiana_pivoteamento_parcial(a,b)
```

[20]:
```
(array([[ 2. ,  1. , -1. ],
        [ 0. ,  1. ,  2. ],
        [ 0. ,  0. ,  0.5]]),
 array([0. , 1. , 0.5]),
 array([ 1., -1.,  1.]),
 0.0)
```

[21]:
```
sla.solve(a,b)
```

[21]:
```
array([ 1., -1.,  1.])
```

O resultado obtido está de acordo com o cálculo análitico

Example 6

$$\begin{bmatrix} \varepsilon & 2 \\ 1 & \varepsilon \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \tag{1}$$

Exercise 5 Verifique o exemplo 6 para valores pequenos de ε, usando as funções com e sem pivotamento parcial.

[22]:
```
e = 1e-1

a = np.array([[e, 2], [1, e]])
b = np.array([4, 3])
```

[23]:
```
elimin_gaussiana(a, b)
```

[23]:
```
(matrix([[  0.1,   2. ],
         [  0. , -19.9]]),
 matrix([[  4.],
         [-37.]]),
 array([[2.81407035],
        [1.85929648]]),
 8.881784197001252e-16)
```

[24]:
```
elimin_gaussiana_pivoteamento_parcial(a,b)
```

[24]:
```
(array([[1.  , 0.1 ],
        [0.  , 1.99]]),
 array([3. , 3.7]),
 array([2.81407035, 1.85929648]),
 0.0)
```

[25]:
```
sla.solve(a,b)
```

[25]:
```
array([2.81407035, 1.85929648])
```

[26]:
```
e = 1e-100
```

[27]:
```
elimin_gaussiana(a, b)
```

[27]:
```
(matrix([[  0.1,   2. ],
         [  0. , -19.9]]),
 matrix([[  4.],
         [-37.]]),
 array([[2.81407035],
        [1.85929648]]),
 8.881784197001252e-16)
```

[28]:
```
elimin_gaussiana_pivoteamento_parcial(a,b)
```

[28]:
```
(array([[1.  , 0.1 ],
        [0.  , 1.99]]),
 array([3. , 3.7]),
 array([2.81407035, 1.85929648]),
 0.0)
```

[29]:
```
sla.solve(a,b)
```

[29]:
```
array([2.81407035, 1.85929648])
```

É possível observar que para o valor de $e = 0{,}1$ houve um dos métodos que apresetaram erro de cálculo, demonstrando assim o efeito do cancelamento catastrófico