Aula 03: Numpy - Parte 1

1- Passos iniciais

Importando o numpy

```
import numpy as np
```

Criando um array:

```
lista = [1,2,3,4,5]
print(lista)
a = np.array(lista)
print(a)
```

```
[1, 2, 3, 4, 5]
[1 2 3 4 5]
```

Verificando as dimensões

```
print(a.shape)
a.dtype
```

```
(5,)
dtype('int64')
```

Criando array multidimensional

```
a = np.array([[1,2,3,4], [5,6,7,8]])
print(a)
print(a.shape)
```

```
[[1 2 3 4]
 [5 6 7 8]]
(2, 4)
```

2. Funções para criação de array

2.1 Função empty: Cria um array da dimensão especificada com valores aleatórios

```
a = np.empty([3,3])
print(a)
```

```
[[4.65484060e-310 3.60739284e-313 1.38338381e-322]
 [4.65484060e-310 0.00000000e+000 0.00000000e+000]
 [4.94065646e-323 0.00000000e+000 0.00000000e+000]]
```

2.2 Função np.zeros: Cria um array preenchido com zeros com as dimensões especificadas

```
a = np.zeros([3,5])
print(a)
print(a.shape)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
(3, 5)
```

2.3 Função np.ones: Cria um array preenchido com um's com as dimensões especificadas

```
a = np.ones([2,10], dtype = int)
print(a)
```

```
print(a.shape)
```

```
[[1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1]]
(2, 10)
```

2.4 Função np.arange: Cria um array preenchido com valores que vão de um número a outro seguindo um determinado passo.

```
a = np.arange(10, 20, 1)
print(a)
b = np.arange(1, 10, 2)
print(b)
```

```
[10 11 12 13 14 15 16 17 18 19]
[1 3 5 7 9]
```

2.5 Função np.linspace: Cria um array preenchido com valores igualmente espaçados dentro de um determinado intervalo

```
a = np.linspace(1, 2, 5)
print(a)
```

```
[1.   1.25 1.5  1.75 2.  ]
```

2.6 Função np.astype: Converte um determinado tipo de dado de um array em outro (não altera o original)

```
b = a.astype(int)
print(b)
```

```
[1 1 1 1 2]
```

3. Slice de array ou indexação

```
# array unidimensional
a = np.array([11, 32, 56, 84])
print(a[0])
print(a[3])
print(a[0:2])
print(a[-1])
```

```
11
84
[11 32]
84
```

```
# array bidimensional
a = np.array([[11, 32, 56, 84], [35, 56, 87, 91]])
print(a[:, :-1])
print(a[:, 0])
```

```
[[11 32 56]
 [35 56 87]]
[11 35]
```

3.1 Exemplo prático de como usa-se slice de array para criação de conjuntos de treino e teste

```
data = np.array([[11, 32, 46, 56], [1, 30, 95, 33], [10, 100, 21, 13]])
split = 2
train, test = data[:split, :], data[split:, :]
print(data)
print("Conjunto treino:\n", train)
print("Conjunto teste:\n", test)
```

```
[[ 11  32  46  56]
 [  1  30  95  33]
 [ 10 100  21  13]]
Conjunto treino:
```

```
 [[11 32 46 56]
 [ 1 30 95 33]]
Conjunto teste:
 [[ 10 100  21  13]]
```

3.2 Indexação booleana

```
a = np.linspace(0, 100, 100, dtype = int)
x = a[(a > 20) & ( a > 50)] # Maiores que 20 e maiores que 50 (ou seja, maiores que 50)
print(x)
y = a[(a < 10) | (a > 90)] # Menores que 10 ou maiores que 90
print(y)
```

```
[ 51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86
  87  88  89  90  91  92  93  94  95  96  97  98 100]
[  0   1   2   3   4   5   6   7   8   9  91  92  93  94  95  96  97  98
 100]
```

4- Shape de dados

```
a = np.arange(1,51)
print(a)
print(a.shape)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 49 50]
(50,)
```

```
data = np.array([[11,32,67],
                 [6,43,21],
                [32,54,61],
                 [21,87,45]])
print(data.shape)
print("Linhas = %d" % data.shape[0])
print("Colunas = %d" % data.shape[1])
```

```
(4, 3)
Linhas = 4
Colunas = 3
```

4.1 Função Reshape

Depois de fatiar seus dados, pode ser necessário redimensioná-los.

Por exemplo, algumas bibliotecas, como scikit-learn, podem exigir que um array unidimensional de variáveis de saída (y) seja moldados como um array bidimensional com uma coluna e resultados para cada coluna.

Alguns algoritmos, como a redes neurais recorrentes LSTM (Long Short-Term Memory) em Keras, requer que a entrada seja especificada como uma matriz tridimensional composta de amostras, etapas de tempo e recursos.

É importante sabermos redimensionar seus arrays NumPy para que seus dados atendam à expectativa de Bibliotecas Python.

Transformando vetor 1D em 2D

```
a = np.arange(6)
print(a)
print("Redimensionado para 2D:")
a = a.reshape(2,3)
print(a)
print("Redimensionado para 2D:")
a = a.reshape(3,2)
print(a)
```

```
[0 1 2 3 4 5]
Redimensionado para 2D:
[[0 1 2]
 [3 4 5]]
Redimensionado para 2D:
[[0 1]
 [2 3]
 [4 5]]
```

Transformando vetor 2D em 3D

```
a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])
a.reshape(a.shape[0], a.shape[1], 1)
```

```
array([[[ 1],
        [ 2],
        [ 3],
        [ 4],
        [ 5]],

       [[ 6],
        [ 7],
        [ 8],
        [ 9],
        [10]],

       [[11],
        [12],
        [13],
        [14],
        [15]]])
```

4.2 Função flatten

A função flatten transforma qualquer vetor ndimensional em uma só dimensão.

```
data = np.array([[11,32,67],
                 [6,43,21],
                [32,54,61],
                 [21,87,45]])
data = data.flatten()
print(data)
print(data.shape)
```

```
[11 32 67  6 43 21 32 54 61 21 87 45]
(12,)
```

4.3 Função insert

A função insert insere um ou mais valores em uma determinada linha ou coluna de um vetor

```
data = np.array([[5, 5, 5], [5, 5, 5], [5, 5, 5]])
print(data)
# inserindo uma nova coluna na posição 1
data = np.insert(data, 1, [0, 0, 0], axis = 1)
print(data)
```

```
[[5 5 5]
 [5 5 5]
 [5 5 5]]
[[5 0 5 5]
 [5 0 5 5]
 [5 0 5 5]]
```

```
data = np.array([[5, 5, 5], [5, 5, 5], [5, 5, 5]])
print(data)
# inserindo uma nova linha
data = np.insert(data, 2, [1, 2, 3], axis = 0)
print(data)
```

```
[[5 5 5]
 [5 5 5]
 [5 5 5]]
[[5 5 5]
 [5 5 5]
 [1 2 3]
 [5 5 5]]
```

4.4 Função delete

```
#np.delete(array, posição, axis)
data = np.array([[11,32],
```

```
              [6,43],
              [32,7]])

#remove a primeira linha
data = np.delete(data,0,0)
print(data)
```

```
     [[ 6 43]
      [32  7]]
```

Exercícios

Exercício 1:

Crie um array com 50 elementos (começando do 1) e depois selecione apenas os múltiplos de 3

```
a = np.arange(1, 51)
a[a % 3 == 0]
```

```
     array([ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48])
```

Exercício 2:

1- Crie um array de 10 posições:

2- insira 3 elementos no início desse array

3- Remova o último elemento do array

4- Redimensione o array como uma matriz 3x4

5- Insira uma nova linha com apenas zeros no final da matriz

6- Exiba a dimensionalidade da matriz final

```
# 1
a = np.linspace(0, 10, 10, dtype = int, endpoint = False)
print(a)
```

```
     [0 1 2 3 4 5 6 7 8 9]
```

```
# 2

for i in range(3):
  a = np.insert(a, 0, 0)
# Ao invés de usar o loop, pode-se adicionar os três elementos passando um vetor como argumento nessa função: a = np.insert(a
print(a)
print(a.shape)
```

```
     [0 0 0 0 1 2 3 4 5 6 7 8 9]
     (13,)
```

```
# 3
print(len(a))
a = a[: len(a) - 1]
#Também é possível usar a função delete: a = np.delete(a, (len(a) - 1))
print(len(a))
```

```
     13
     12
```

```
# 4
a = np.reshape(a, (3, 4))
print(a)
a.shape
```

```
     [[0 0 0 0]
      [1 2 3 4]
      [5 6 7 8]]
     (3, 4)
```

```
# 5 e 6
a = np.insert(a, len(a), [0, 0, 0, 0], axis = 0)
print(a)
a.shape
```

```
[[0 0 0 0]
 [1 2 3 4]
 [5 6 7 8]
 [0 0 0 0]]
(4, 4)
```

Aula 04: Numpy - Parte 02

6. Operações sobre Arrays

max(): Retorna o valor máximo de um array

```
a = np.random.rand(3)
print(a)
a.max()
```

```
[0.75225797 0.65594001 0.39247899]
0.7522579659932921
```

min(): Retorna o valor mínimo de um array

```
a = np.random.rand(3)
print(a)
a.min()
```

```
[0.65116117 0.01231949 0.14555125]
0.012319493604852738
```

argmax(): Retorna o índice do valor máximo de um array

```
a = np.random.randint(0, 10, 5)
print(a)
print(a.argmax())
```

```
[4 2 6 4 5]
2
```

argmin(): Retorna o índice do valor mínimo de um array

```
a = np.random.rand(10)
print(a)
print(a.argmin())
```

```
[0.2040261  0.21379574 0.86770501 0.05288594 0.295702   0.32359513
 0.23760663 0.5198658  0.84410515 0.41246346]
3
```

mean(): Retorna o valor que representa a média dos elementos de um array

```
a = np.random.randint(0, 100, 5)
print(a)
a.mean()
```

```
[ 6 14 63 72 10]
33.0
```

unique():

```
a = np.random.randint(0, 7, 10)
print(a)
np.unique(a) #Retorna os elementos únicos do array a
```

```
[6 2 5 0 3 3 3 1 6 5]
array([0, 1, 2, 3, 5, 6])
```

max(): Retorna o valor máximo de um array

```
a = np.random.rand(3)
print(a)
a.max()
```

```
[0.69466555 0.06906879 0.18815736]
0.6946655505327061
```

min(): Retorna o valor mínimo de um array

```
a = np.random.rand(3)
print(a)
a.min()
```

```
[0.37968479 0.97331152 0.8936621 ]
0.37968478720210896
```

argmax(): Retorna o índice do valor máximo de um array

```
a = np.random.randint(0, 10, 5)
print(a)
print(a.argmax())
```

```
[3 0 9 8 5]
2
```

argmin(): Retorna o índice do valor mínimo de um array

```
a = np.random.rand(10)
print(a)
print(a.argmin())
```

```
[0.35203954 0.37485804 0.69136711 0.42924991 0.01993705 0.3978754
 0.59644421 0.91535338 0.25269227 0.17467917]
4
```

mean(): Retorna o valor que representa a média dos elementos de um array

```
a = np.random.randint(0, 100, 5)
print(a)
a.mean()
```

```
[10 54 18 33 77]
38.4
```

unique():

```
a = np.random.randint(0, 7, 10)
print(a)
np.unique(a) #Retorna os elementos únicos do array a
```

```
[1 5 6 1 6 2 0 2 1 2]
array([0, 1, 2, 5, 6])
```

7. Números aleatórios

np.random.randint: Gera números inteiros aleatórios no range especificado

```
a = np.random.randint(0,10)
print(a)
b = np.random.randint(0, 10, 10)
print(b)
```

```
4
```

```
[0 1 8 6 8 3 6 1 0 7]
```

np.random.rand: Gera números reais aleatórios entre 0 e 1

```
a = np.random.rand()
print(a)
b = np.random.rand(10)
print(b)
```

```
0.7851390202266852
[0.15054207 0.08507474 0.95311091 0.1378107  0.55939523 0.77031127
 0.67575186 0.81282276 0.71187258 0.60893834]
```

```
# Criando matrizes com números aleatórios:
c = np.random.rand(5, 5)
print(c)
# Transformando a matriz de reais em inteiros usando a função astype
c = (c*10).astype(int)
print(c)
```

```
[[1.95144231e-01 6.95814021e-01 4.60623517e-01 9.99023066e-02
  3.81270462e-01]
 [4.89060395e-01 2.52518011e-01 5.62007419e-01 6.10677855e-01
  3.46132703e-01]
 [4.73905894e-01 6.96506081e-01 1.48989939e-01 9.35010920e-01
  4.01415900e-04]
 [5.49607987e-02 7.58145704e-01 8.43468633e-01 8.31999929e-01
  3.71578593e-01]
 [1.05967335e-01 9.94708738e-01 4.42295826e-01 2.56022304e-01
  9.45651241e-01]]
[[1 6 4 0 3]
 [4 2 5 6 3]
 [4 6 1 9 0]
 [0 7 8 8 3]
 [1 9 4 2 9]]
```

np.random.randn: Gera uma matriz com as dimensões especificadas a partir de valores de uma distribuição normal padrão (Caso a curva normal não seja a padrão, usar a fórmula *dp * np.random.randn() + média*)

```
a = np.random.randn(3, 3)
a
```

```
array([[ 0.40576456,  0.64944389, -0.10315239],
       [ 0.64235488,  0.99211446, -0.26993654],
       [ 0.18855817,  1.35466215,  0.32037022]])
```

```
# Gerando uma matriz 3x3 a partir de valores de uma distribuição normal de média = 10 e dp = 2 (olhar a fórmula)
a = 2 * np.random.randn(3, 3) + 10
a
```

```
array([[13.9846843 ,  8.58691929,  6.02974624],
       [12.91617969,  8.65667143, 10.99159135],
       [11.95478476, 10.57248194,  6.51818091]])
```

Exercícios

Exercício 3: a) Gere um array de 25 elementos, começando no elemento 1 até o elemento 25 b) Converter em uma matriz 5x5 c) Substituir os elementos pares por 0 e ímpares por 1 d) Calcule quantos elementos ímpares existem em cada linha.

```
#a
a = np.arange(1, 26, dtype = int)
a
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25])
```

```
#b
print(a)
print(a.shape)
a = np.reshape(a, (5,5))
print(a)
print(a.shape)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25]
(25,)
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
(5, 5)
```

```
#c
print(a)
a = np.where(a % 2 == 0, 0, a)
print(a)
a = np.where(a % 2 == 1, 1, a)

# Também é possível resolver da seguinte maneira: a[a % 2 == 0] = 0 e a[a % 2 == 1] = 1
print(a)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]
 [21 22 23 24 25]]
[[ 1  0  3  0  5]
 [ 0  7  0  9  0]
 [11  0 13  0 15]
 [ 0 17  0 19  0]
 [21  0 23  0 25]]
[[1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]
 [0 1 0 1 0]
 [1 0 1 0 1]]
```

```
#d
#Solução manual
contador = np.array([], dtype = int)
for i in a:
  cont = 0
  for j in i:
    if j % 2 == 1:
      cont += 1
  contador = np.insert(contador, 0, cont)
print(contador)

#Solução usando o comando sum
print(a.sum(axis = 1))
```

```
[3 2 3 2 3]
[3 2 3 2 3]
```

Exercício 4: Calcular o volume do cilindro, dado raio = 10 e altura = 50

```
raio = 10
altura = 50

volume = np.pi * (raio ** 2) * altura
print(volume)
```

```
15707.963267948966
```

Exercício 5: Calcular o volume de 5 cilindros, dados raios = [10, 15, 11, 17, 19] alturas = [100, 120, 115, 140, 180]

```
raios = np.array([10, 15, 11, 17, 19])
alturas = np.array([100, 120, 115, 140, 180])
volumes = (raios ** 2) * np.pi * alturas
print(volumes)
```

```
[ 31415.9265359   84823.00164692  43715.2617747  127108.83876424
 204140.69063026]
```

Exercício 6: Utilize o NumPy para calcular a distância Euclidiana entre os vetores x = (2,8) e y = (9,1)

```
x = np.array([2, 8])
y = np.array([9, 1])
```

```
d = np.sqrt(np.sum((x - y)**2))
d
```

```
9.899494936611665
```

Exercício 7: Simule 20 arremessos de um dado de 6 faces • Conte quantas vezes o número 5 apareceu nesses 20 arremessos

```
np.random.seed(1)
tentativas = np.random.choice(a = [1, 2, 3, 4, 5, 6], size = 20, p = [1/6, 1/6, 1/6, 1/6, 1/6, 1/6])
print(tentativas)
tentativas[tentativas != 5] = 0
tentativas[tentativas == 5] = 1
tentativas.sum()
#Esse método altera o vetor original
```

```
[3 5 1 2 1 1 2 3 3 4 3 5 2 6 1 5 3 4 1 2]
3
```

Exercício 8: Construir uma tabela que representa os dados de um grupo de 10 pessoas. Cada pessoa será representada como uma linha da tabela, e as características de cada pessoa são descritas nas colunas. Considerar as seguintes colunas:

- Idade: Média 30.0, desvio padrão 5
- Peso: Média 60.0, desvio padrão 3
- Altura: Média 1.70, desvio padrão 0.20
- Salario: Média 2000.0, desvio padrão 500

• Após a construção da tabela, calcule e exiba o valor máximo e o mínimo de cada característica

```
# Ajustando configurações de ponto flutuante e exibição de notação científica
np.set_printoptions(precision = 2, suppress = True)

#Construindo as colunas de acordo com as distribuições informadas:
idade = 5 * np.random.randn(10) + 30
peso = 3 * np.random.randn(10) + 60
altura = 0.2 * np.random.randn(10) + 1.7
salario = 500 * np.random.randn(10) + 2000

# Inserindo os vetores criados como colunas em um array pessoas
pessoas = idade.reshape(10, 1)
print("idade\n",pessoas)
pessoas = np.insert(pessoas, 1, peso, axis = 1)
print("idade, peso\n",pessoas)
pessoas = np.insert(pessoas, 2, altura, axis = 1)
print("idade, peso, altura\n",pessoas)
pessoas = np.insert(pessoas, 3, salario, axis = 1)
print("idade, peso, altura, salario\n",pessoas)

# Calculando o valor máximo e mínimo de cada característica:
print("Valores máximos\n",pessoas.max(axis = 0))
print("Valores mínimos\n",pessoas.min(axis = 0))
```

```
#Adicional: Criar conjunto de treino e conjunto de teste
split = 8
treino = pessoas[:split, :]
teste = pessoas [split:, :]
print("Conjunto treino\n",treino)
print("Conjunto teste\n",teste)
```

```
idade
 [[35.67]
 [24.5 ]
 [29.14]
 [25.61]
 [30.21]
 [32.91]
 [24.5 ]
 [35.72]
 [34.51]
 [32.51]]
idade, peso
 [[35.67 62.7 ]
 [24.5  57.95]
 [29.14 59.63]
 [25.61 57.19]
 [30.21 59.2 ]
 [32.91 61.59]
 [24.5  57.93]
 [35.72 58.81]
 [34.51 57.94]
```

```
 [32.51 57.46]]
idade, peso, altura
 [[35.67 62.7   1.57]
 [24.5  57.95  1.7 ]
 [29.14 59.63  1.48]
 [25.61 57.19  1.75]
 [30.21 59.2   2.03]
 [32.91 61.59  1.85]
 [24.5  57.93  1.66]
 [35.72 58.81  1.52]
 [34.51 57.94  1.55]
 [32.51 57.46  2.04]]
idade, peso, altura, salario
 [[  35.67   62.7     1.57 2025.4 ]
 [  24.5    57.95    1.7  1681.5 ]
 [  29.14   59.63    1.48 2095.46]
 [  25.61   57.19    1.75 3050.13]
 [  30.21   59.2     2.03 2060.08]
 [  32.91   61.59    1.85 2308.6 ]
 [  24.5    57.93    1.66 2150.09]
 [  35.72   58.81    1.52 1823.88]
 [  34.51   57.94    1.55 1428.74]
 [  32.51   57.46    2.04 1825.33]]
Valores máximos
 [  35.72   62.7     2.04 3050.13]
Valores mínimos
 [  24.5    57.19    1.48 1428.74]
Conjunto treino
 [[  35.67   62.7     1.57 2025.4 ]
 [  24.5    57.95    1.7  1681.5 ]
 [  29.14   59.63    1.48 2095.46]
 [  25.61   57.19    1.75 3050.13]
 [  30.21   59.2     2.03 2060.08]
 [  32.91   61.59    1.85 2308.6 ]
 [  24.5    57.93    1.66 2150.09]
 [  35.72   58.81    1.52 1823.88]]
Conjunto teste
```

Aula 05 - Espaços Vetoriais

Aula teórica (espaço vetorial)

Aula 06 - Bases Vetoriais

Aula teórica (Base vetorial)

Ver Revisão: Espaço Vetorial e Base Vetorial (aula 07)

Aula 07: Matrizes - parte 1

Revisão: Espaço Vetorial e Base Vetorial

Espaço Vetorial

Um espaço vetorial V consiste em um conjunto não vazio de objetos que podem ser somados e multiplicados por um número que pertence a um conjunto determinado, satisfazendo as seguintes propriedades de soma e multiplicação:

Soma:

1. $\vec{u}$,$\vec{v}$ ∈ V: $\vec{u}$ + $\vec{v}$ ∈ V
2. Propriedade comutativa: $\vec{u}$ + $\vec{v}$ = $\vec{v}$ + $\vec{u}$
3. Propriedade associativa: ($\vec{u}$ + $\vec{v}$) + $\vec{w}$ = ($\vec{w}$ + $\vec{v}$) + $\vec{u}$
4. Elemento neutro: ∃ $\vec{0}$ ∈ V | $\vec{v}$ + $\vec{0}$ = $\vec{v}$ ∀ $\vec{v}$
5. Elemento oposto: $\vec{v}$ + ($\overrightarrow{-v}$) = $\vec{0}$

Multiplicação

1. x_1 ∈ V, x ∈ U | $\vec{v}$ * x ∈ V
2. Propriedade distributiva: x * ($\vec{v}$ + $\vec{w}$) = x * $\vec{v}$ + x * $\vec{w}$

3. Propriedade associativa: $\vec{v}$ (x_1 * x_2) = x_1 ($\vec{v}$ * x_2)
4. 1 * $\vec{v}$ = $\vec{v}$

Combinação Linear

Definição: É o ato de gerar um vetor à partir de outros usando pesos (esses pesos na verdade são chamados de constantes, e são escalares que multiplicam os vetores)

Por exemplo: Dados $\overrightarrow{v_1}$, $\overrightarrow{v_2}$ e $\overrightarrow{v_3}$, obtenha vetores $\vec{y}$ à partir de uma combinação linear.

Solução:

CL1: $\vec{y}$ = 2$\overrightarrow{v_1}$ + 3$\overrightarrow{v_2}$ + $\overrightarrow{v_3}$

CL2: $\vec{y}$ = 5$\overrightarrow{v_1}$ - 9$\overrightarrow{v_2}$ + 10$\overrightarrow{v_3}$

$$CL_n : \vec{y} = C_1 * \overrightarrow{v_1} + C_2 * \overrightarrow{v_2} + C_3 * \overrightarrow{v_3} + \ldots + C_n * \overrightarrow{v_n}$$

Obs: Os pesos (nesse caso C_1, C_2 e C_3) que multiplicam os vetores pré-estabelecidos da combinação linear devem sempre ser escalares.

Exercício: O vetor $\vec{v} = [8, -2]$ é combinação linear de $\overrightarrow{v_1} = [1, 1]$ e $\overrightarrow{v_2} = [1, -1]$

Solução:

Utilizando a "fórmula" geral apresentada acima...

$CL_n : \vec{y} = C_1 * \overrightarrow{v_1} + C_2 * \overrightarrow{v_2} + C_3 * \overrightarrow{v_3} + \ldots + C_n * \overrightarrow{v_n}$

Temos a seguinte equação:

$$\begin{bmatrix} 8 \\ -2 \end{bmatrix} = C_1 * \begin{bmatrix} 1 \\ 1 \end{bmatrix} + C_2 * \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Caímos em um sistema linear:

$$\begin{cases} C_1 + C_2 = 8 \\ C_1 - C_2 = -2 \end{cases}$$

Resolvendo o sistema linear, temos que C_1 = 3, C_2 = 5. Esses são os pesos para que a combinação linear de $\overrightarrow{v_1}$ e $\overrightarrow{v_2}$ resulte no vetor $\vec{v}$

Dependência Linear

Vetores linearmente dependentes: São vetores que podem ser obtidos através de uma combinação linear de vetores do conjunto V, enquanto vetores linearmente independentes não podem ser obtidos através de combinação linear de vetores de V.

Na prática, vetores linearmente dependentes são paralelos (possuem a mesma direção), enquanto vetores linearmente independentes não são paralelos.

Como saber se dois ou mais vetores são L.I. (linearmente independentes)?

- **Método 1:** Se a C.L. desses vetores em um vetor nulo tiver como resultado das constantes **apenas** a solução trivial ($C_1 = C_2 = C_3 = \ldots = C_n = 0$)
- **Método 2:** Caso **não** haja uma razão constante entre cada componente dos vetores.
  Por exemplo, os vetores $\overrightarrow{v_1} = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix}$ e $\overrightarrow{v_2} = \begin{bmatrix} 2 \\ 10 \\ 14 \end{bmatrix}$ são L.D. pois possuem uma mesma razão r entre cada elemento correspondente (r = 1/2 = 5/10 = 7/14)
- **Método 3:** Por último, o determinante da matriz composta pelos vetores analisados deve ser $\neq 0$

Base vetorial

Um conjunto de vetores {$\overrightarrow{v_1}$, $\overrightarrow{v_2}$, $\overrightarrow{v_3}$, ..., $\overrightarrow{v_n}$} pertencentes ao espaço vetorial V será chamado de base de V, se respeitarem as seguintes propriedades:

- I- **{$\overrightarrow{v_1}$, $\overrightarrow{v_2}$, $\overrightarrow{v_3}$, ..., $\overrightarrow{v_n}$} são L.I. (Linearmente Independentes)**

- II- **[$\overrightarrow{v_1}$, $\overrightarrow{v_2}$, $\overrightarrow{v_3}$, ..., $\overrightarrow{v_n}$] é conjunto gerador de V**
- III- **Os vetores do conjunto precisam ter a mesma dimensão e o número de vetores do conjunto deve ser igual ou superior ao número de componentes (dimensão) dos vetores.**

Obs: O uso de chaves na segunda propriedade indica obrigatoriamente CONJUNTO GERADOR, que é o nome dado ao conjunto de vetores que consegue, através de CL, gerar qualquer vetor de V.

Exercício 1: O conjunto {$\overrightarrow{e_1}$, $\overrightarrow{e_2}$}, com $\overrightarrow{e_1}$ = (1, 0) e $\overrightarrow{e_2}$ = (0, 1) é uma base de $\mathbb{R}^2$. Essa base é chamada de base canônica de $\mathbb{R}^2$

Resposta:
Para que o conjunto apresentado seja de fato uma base de $\mathbb{R}^2$, as propriedades vistas devem ser confirmadas:

Propriedade I- Para checar se são linearmente independentes, uma das formas vistas é verificar se o determinante da matriz composta pelos vetores seja diferente de zero:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

det(A) = 1 - 0 = 1, logo os vetores da base canônica do espaço vetorial apresentado ($\mathbb{R}^2$) são L.I. (linearmente independentes)

Propriedade II- Para checar se é conjunto gerador (ou seja, podem formar outros vetores através de combinações lineares), utilizemos uma forma genérica de representar um vetor qualquer de $\mathbb{R}^2$, $\vec{v} = [x, y]$. Precisamos verificar se a base canônica é geradora de $\vec{v}$. Para isso utilizamos a fórmula vista da combinação linear:

$$\begin{bmatrix} x \\ y \end{bmatrix} = C_1 . \begin{bmatrix} 1 \\ 0 \end{bmatrix} + C_2 . \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\begin{cases} C_1 + 0 = x \\ C_2 + 0 = y \end{cases} \iff C_1 = x, C_2 = y$$

Vemos que os pesos dos vetores da base são respectivamente iguais a x e y. Logo, qualquer vetor contido em $\mathbb{R}^2$ pode ser obtido através de C.L. dos vetores da base canônica (em $\mathbb{R}^2$, são $[0, 1]$ e $[1, 0]$).

Propriedade III- Os vetores possuem a mesma dimensão (ambos são bidimensionais) e o número de vetores contidos na base é igual à dimensionalidade dos vetores.

Por isso, ao confirmar as propriedades, podemos dizer que o conjunto $[[1, 0], [0, 1]]$ é gerador e, portanto, é base vetorial de $\mathbb{R}^2$

Exercício 2: Verifique que o conjunto {$\overrightarrow{v_1}$, $\overrightarrow{v_2}$}, com $\overrightarrow{v_1}$ = (1, 1) e $\overrightarrow{v_2}$ = (2, -1) é uma base de $\mathbb{R}^2$\

Resposta: Assim como no exercício 1, checamos as propriedades de bases vetoriais:

- **Propriedade 1:**
  $$A = \begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}$$
  det(A) = -1 -2 = -3
  Determinante da matriz é diferente de zero, então são linearmente independentes
- **Propriedade 2:**
  $$\begin{bmatrix} x \\ y \end{bmatrix} = C_1 . \begin{bmatrix} 1 \\ 1 \end{bmatrix} + C_2 . \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

$$\begin{cases} C_1 + 2C_2 = x \\ C_1 - C_2 = y \end{cases} \iff C_1 = \frac{x+2y}{3}, C_2 = \frac{x-y}{3}$$

Como existem valores em $\mathbb{R}^2$ que satisfazem as constantes, então dizemos que o conjunto é gerador, portanto uma base vetorial de $\mathbb{R}^2$

- **Propriedade 3:**
  Também está satisfeita pois existem dois vetores no conjunto gerador (ambos bidimensionais)

Conteúdo da Aula:

Resolvendo sistemas lineares utilizando matrizes:
É necessário que a matriz seja quadrática (1x1, 2x2, 3x3, 4x4, ...) e o seu determinante seja diferente de zero.

Para resolver o sistema, basta calcular cada variável v como det(v) / det(m). Exemplo:

Tenhamos o seguinte sistema linear:

$$\begin{cases} x + y + z = 6 \\ x + 2y + 2z = 9 \\ 2x + y + 3z = 11 \end{cases}$$

O valor de x, por exemplo, será expresso por det(x)/det(m), sendo:

m a matriz composta pelos coeficientes das variáveis do sistema linear (aqui são os valores que acompanham x, y e z);

x a matriz composta pelos coeficientes das variáveis y e z junto aos resultados das equações substituindo onde estariam os coeficientes de x.

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

$$X = \begin{bmatrix} 6 & 1 & 1 \\ 9 & 2 & 2 \\ 11 & 1 & 3 \end{bmatrix}$$

x, portanto será expresso pelo determinante da matriz x dividido pelo determinante da matriz M, ou seja, x = $\frac{det(x)}{det(m)}$

O mesmo raciocínio vale para as variáveis restantes do sistema linear.

Nesse caso, y = $\frac{det(y)}{det(m)}$ e z = $\frac{det(z)}{det(m)}$

Obs: A título de consulta, a matriz y e a matriz z são as seguintes

$$Y = \begin{bmatrix} 1 & 6 & 1 \\ 1 & 9 & 2 \\ 2 & 11 & 3 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 1 & 6 \\ 1 & 2 & 9 \\ 2 & 1 & 11 \end{bmatrix}$$

Definição de Vetor

```
#Um vetor é representado em Python como um array da Numpy
v = np.array([1,2,3])
print(v)
```

```
[1 2 3]
```

Operações com Vetor

```
# Soma de vetores
v, u = np.array([1,2,3]), np.array([4,5,6])
soma = v + u
print(soma)
```

```
[5 7 9]
```

```
#Subtração de vetores
sub = v - u
print(sub)
```

```
[-3 -3 -3]
```

```
# Multiplicação de vetores
mult = v * u
print(mult)
```

```
[ 4 10 18]
```

```
# Divisão de vetores
div = v / u
print(div)
```

```
[0.25 0.4  0.5 ]
```

```
# Produto interno (somatório da multiplicação de componente a componente dos vetores)
prod_int = np.dot(v, u)
print(prod_int)
```

```
#Outras maneiras de escrever esse produto interno entre v e u:
print('método 2: {}'.format(v.dot(u)))
print('método 3: {}'.format(u.dot(v)))
```

```
32
método 2: 32
método 3: 32
```

```
# Multiplicação de vetor por escalar
print(u)
print(3 * u)
```

```
[4 5 6]
[12 15 18]
```

```
# Norma L1 do vetor (somatório da norma de cada componente do vetor)
# L1 de |v| = |v1| + |v2| + |v3| + ... + |vn|
print(u)
print(np.linalg.norm(u, 1))
```

```
[4 5 6]
15.0
```

```
# Norma L2 do vetor, norma Euclidiana. É a padrão dessa função
# L2 de |v| = sqrt(|v1|**2 + |v2|**2 + |v3|**2 + ... + |vn|**2)
print(u)
print(np.linalg.norm(u))
```

```
[4 5 6]
8.774964387392123
```

```
# Norma máxima do vetor
# Linf(v) de |v| = |v|inf = max{v1, v2, v3}
import math
print(u)
print(np.linalg.norm(u, math.inf))

# Norma mínima do vetor
# Linf(v) de |v| = |v|inf = max{v1, v2, v3}
print(np.linalg.norm(u, - math.inf))
```

```
[4 5 6]
6.0
4.0
```

```
# Infinito em Python
inf_pos = float('inf')
inf_neg = float('-inf')
print('Infinito positivo: {}'.format(inf_pos))
print('Infinito negativo: {}'.format(inf_neg))
print(inf_pos > 5)
print(inf_pos < 1)
```

```
Infinito positivo: inf
Infinito negativo: -inf
True
False
```

Definição de Matriz

```
# Uma matriz pode ser representada em Python usando array 2D
# Pode ser construído por uma lista de lista
A = np.array([
    [1,2,3],
    [4,5,6]]
)
print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

Operação com Matrizes

```
# Soma de Matrizes:
A = np.array([
              [1,2,3],
              [4,5,6]
])
B = np.array([
              [7,8,9],
              [10,11,12]
])
C = A + B
print('soma:\n',C)
# Subtração de matrizes
D = A - B
print('subtração:\n',D)
```

```
soma:
 [[ 8 10 12]
 [14 16 18]]
subtração:
 [[-6 -6 -6]
 [-6 -6 -6]]
```

```
#Multiplicação de matrizes (Produto de Hadamard).
# É a multiplicação de cada componente de uma matriz pela correspondente
# em outra matriz:
print('A:\n',A)
print()
print('B:\n',B)
print()
Mult_Matriz = A * B
print('Produto de Hadamard:\n',Mult_Matriz)
```

```
A:
 [[1 2 3]
 [4 5 6]]

B:
 [[ 7  8  9]
 [10 11 12]]

Produto de Hadamard:
 [[ 7 16 27]
 [40 55 72]]
```

```
# Divisão de matrizes.
# Divisão de cada componente de uma matriz pela correspondente
# em outra matriz:
print('A:\n',A)
print()
print('B:\n',B)
print()
Div_Matriz = A / B
print('Divisão:\n',Div_Matriz)
```

```
A:
 [[1 2 3]
 [4 5 6]]

B:
 [[ 7  8  9]
 [10 11 12]]

Divisão:
 [[0.14 0.25 0.33]
 [0.4  0.45 0.5 ]]
```

```
# Multiplicação de matrizes tradicional
# Pré-requisito: N de colunas da primeira == N de linhas da segunda
# ou... len(A[0]) == len(B)
# ou ainda... A(i,j) B(j,k)
A = np.array([[1,2],
              [3,4],
              [5,6]

])
B = np.array([[1,2],
              [3,4]

])
print('A:\n',A)
```

```
print()
print('B:\n',B)
print()
Mult_matrizes = np.dot(A,B) # Método 1
print('Multiplicação tradicional (Método 1, np.dot(A,B)):\n', Mult_matrizes)
print('Multiplicação tradicional (Método 2, operador A@B):\n', A@B)
```

```
A:
 [[1 2]
 [3 4]
 [5 6]]

B:
 [[1 2]
 [3 4]]

Multiplicação tradicional (Método 1, np.dot(A,B)):
 [[ 7 10]
 [15 22]
 [23 34]]
Multiplicação tradicional (Método 2, operador A@B):
 [[ 7 10]
 [15 22]
 [23 34]]
```

```
# Multiplicação de matriz por vetor.
# Pré-requisito: Número de colunas da matriz deve ser igual
# ao número de componentes do vetor.
# O resultado é sempre um vetor
A = np.array([[1,2,3],
              [3,4,5]])

B = np.array([1,2,3])
print('A:\n',A)
print()
print('B:\n',B)
print()
Mult_matriz_vetor = np.dot(A,B) # Método 1
print('Multiplicação tradicional (Método 1, np.dot(A,B)):\n', Mult_matriz_vetor)
print('Multiplicação tradicional (Método 2, operador A@B):\n', A@B)
```

```
A:
 [[1 2 3]
 [3 4 5]]

B:
 [1 2 3]

Multiplicação tradicional (Método 1, np.dot(A,B)):
 [14 26]
Multiplicação tradicional (Método 2, operador A@B):
 [14 26]
```

Tipos de Matrizes

(1) Matriz quadrada:

linhas = colunas

(2) Matriz simétrica:

$A = A^t$

aij = aji

(3) Matriz triangular:

(3.1) Matriz triangular Superior (np.tril(matrix))

(3.2) Matriz triangular Inferior (np.triu(matrix))

```
# Exemplo de matriz triangular:
A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
print('Matriz A:\n', A)
print()
print('Matriz triangular superior:\n', np.triu(A))
print()
print('Matriz triangular inferior:\n', np.tril(A))

# Obs: Os métodos tril e triu não decompõem a matriz A, ou seja,
# essas matrizes triangulares não são as triangulares inferior/
# superior de A (pois esse processo envolve fatoração de matrizes)
```

```
Matriz A:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Matriz triangular superior:
 [[1 2 3]
 [0 5 6]
 [0 0 9]]

Matriz triangular inferior:
 [[1 0 0]
 [4 5 0]
 [7 8 9]]
```

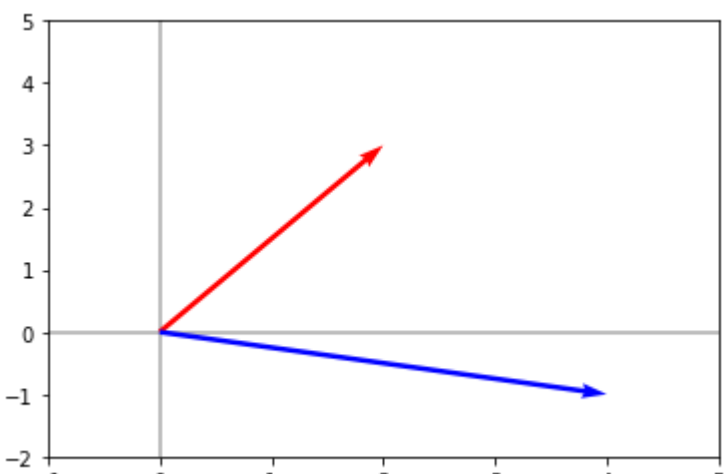
Aula 08: Matrizes - parte 2

Gráfico para vetores

```
import matplotlib.pyplot as plt
```

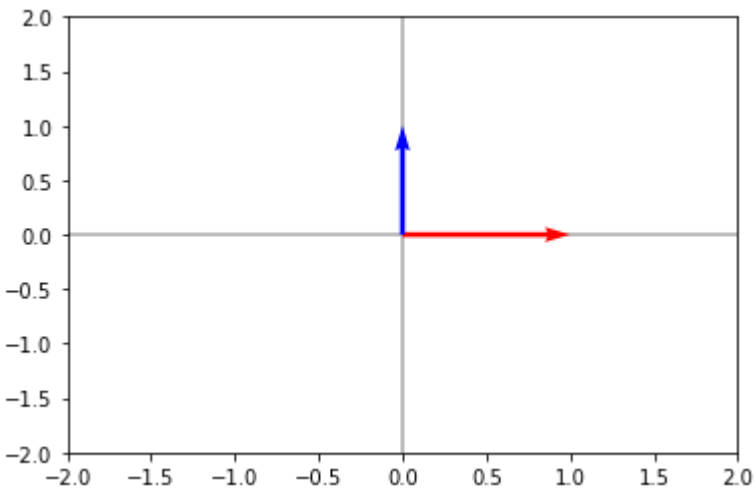
```
def plotVectors(vets, cores, alpha=1):
  plt.figure()
  plt.axvline(x=0, color='#A9A9A9', zorder=0)
  plt.axhline(y=0, color='#A9A9A9', zorder=0)
  for i in range(len(vets)):
    x = np.concatenate([[0,0],vets[i]])
    plt.quiver([x[0]],
               [x[1]],
               [x[2]],
               [x[3]],
               angles ='xy', scale_units='xy', scale=1, color=cores[i], alpha=alpha)


plotVectors([[2,3],[4,-1]], ['red', 'blue'])
plt.xlim(-1, 5)
plt.ylim(-2, 5)
```

```
(-2.0, 5.0)
```




```
u=[1,0]
v=[0,1]
plotVectors([u,v],['red','blue'])
plt.xlim(-2,2)
plt.ylim(-2,2)
```

```
(-2.0, 2.0)
```



Ângulo entre vetores

```
# Produto interno de 2 vetores ortogonais ou perpendiculares (R2) é igual a 0
u = np.array([1, 0])
v = np.array([0, 1])
pInterno = u.dot(v)
print(pInterno)
```

```
0
```

```
# Angulo entre 2 vetores. Calcular arco cosseno do ângulo theta
# theta = arccos(produto_interno / (norma(u) * norma(v)))
def ang2Vetores(u, v):
  pInterno = u.dot(v)
  nu = np.linalg.norm(u)
  nv = np.linalg.norm(v)
  theta = np.arccos(pInterno / (nu * nv)) #valor em radianos
  return np.rad2deg(theta)
```

```
u2 = np.array([1,0])
v2 = np.array([0,1])
print(ang2Vetores(u2, v2))
```

```
90.0
```

Matriz identidade

```
#Para gerar a matriz identidade
I = np.identity(3)
I
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
I2 = np.eye(2, dtype=int)
I3 = np.eye(4, dtype=int)
I4 = np.eye(2, 1, dtype=int)
print(I2)
print()
print(I3)
print()
print(I4)
```

```
[[1 0]
 [0 1]]

[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]

[[1]
 [0]]
```

Matriz transposta

```
A = np.array([[1,2],
              [3,4],
              [5,6]])
```

```
#1 Usando a função transpose() da Numpy
AT = np.transpose(A)
print(AT)
```

```
[[1 3 5]
 [2 4 6]]
```

```
#2 Usando o atributo .T da matriz
AT2 = A.T
print(AT2)
```

```
[[1 3 5]
```

```
[2 4 6]]
```

Determinante

Matriz quadrada A notação: det(A) ou |A|

```
A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
detA = np.linalg.det(A)
print(detA)
```

```
0.0
```

Matriz identidade

Matriz Inversa

A.B = B.A = I

B = A^{-1} (Ou seja, B é inversa de A)

Existe B se, e somente se, det(A) != 0

```
A = np.array([[1.0, 2.0],
              [3.0, 4.0]])
#Condição 1
print('det(A): {}'.format(np.linalg.det(A)))

B = np.linalg.inv(A)
print('A inversa de A:\n{}'.format(B))
```

```
det(A): -2.0000000000000004
A inversa de A:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

```
print('A.B = I:\n{}'.format(A.dot(B)))
```

```
A.B = I:
[[1. 0.]
 [0. 1.]]
```

Matriz Ortogonal

Uma matriz é ortogonal quando sua transposta é igual a sua inversa:

$A^T = A^{-1}$

Sabemos que a inversa de uma matriz existe quando o determinante é diferente de zero

Propriedades:

1. A^{-1} também será ortogonal
2. det(A) = +- 1
3. Dois vetores são ortogonais, quando o produto interno é igual a zero.
4. $A^T.A = A.A^T = I$

```
Q = np.array([[1, 0],
              [0, -1]])
print('Matriz ortogonal Q:\n{}'.format(Q))
print()
print('Matriz inversa Q:\n{}'.format(np.linalg.inv(Q)))
print()
print('Matriz transposta de Q:\n{}'.format(Q.T))
print()
print('Q.QT = I:\n{}'.format(np.dot(Q, Q.T))) #Propriedade 4
```

```
Matriz ortogonal Q:
[[ 1  0]
 [ 0 -1]]

Matriz inversa Q:
[[ 1.  0.]
```

```
 [-0. -1.]]

Matriz transposta de Q:
[[ 1  0]
 [ 0 -1]]

Q.QT = I:
[[1 0]
 [0 1]]
```

Matrizes Esparsas

```
from scipy import sparse as sps
# Criando uma matriz densa
A = np.array([[1, 0, 0, 0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 0, 2, 0, 0, 0],
              [0, 0, 0, 0, 0, 0, 0, 2, 0]])

# Convertendo para uma matriz esparsa (método CSR)
S = sps.csr_matrix(A)
print(S)

#Reconstruindo a matriz densa
B = S.todense()
print(B)

# O número de elementos diferentes de zero
esparsidade = 1.0 - np.count_nonzero(A)/A.size
print('Esparsidade = {}]'.format(esparsidade))
```

```
  (0, 0)        1
  (0, 6)        1
  (1, 5)        2
  (2, 7)        2
[[1 0 0 0 0 0 1 0 0]
 [0 0 0 0 0 2 0 0 0]
 [0 0 0 0 0 0 0 2 0]]
Esparsidade = 0.8518518518518519]
```

Tensor

Um tensor é uma generalização de vetores e matrizes e pode ser entendido como um array multidimensional.

Em Python, um tensor pode ser definido como uma lista de listas, ou array ndimensional

```
T1 = np.array([
              [[1,2,3],[4,5,6],[7,8,9]],
              [[10,11,12], [13,14,15], [16,17,18]],
              [[19,20,21], [22,23,24], [25,26,27]]
              ])
print(T1.shape)
```

```
(3, 3, 3)
```

```
#Adição de T1 com T2
T2 = np.array([[[1, 1, 6],
        [1, 0, 1],
        [6, 9, 1]],

       [[3, 8, 7],
        [5, 7, 4],
        [5, 2, 8]],

       [[5, 8, 6],
        [7, 4, 4],
        [5, 7, 0]]])
print('Soma:\n{}'.format(T1 + T2))
print()

# Produto de tensores (Multiplicação de T1 com T2)
print('Multiplicação:\n{}'.format(T1 * T2))
print()

# Divisão de tensores (Divisão de T1 por T2)
print('Divisão:\n{}'.format(T1 / T2))
```

```
Soma:
[[[ 2  3  9]
  [ 5  5  7]
  [13 17 10]]

 [[13 19 19]
  [18 21 19]
  [21 19 26]]

 [[24 28 27]
  [29 27 28]
  [30 33 27]]]

Multiplicação:
[[[  1   2  18]
  [  4   0   6]
  [ 42  72   9]]

 [[ 30  88  84]
  [ 65  98  60]
  [ 80  34 144]]

 [[ 95 160 126]
  [154  92  96]
  [125 182   0]]]

Divisão:
[[[1.   2.   0.5 ]
  [4.    inf 6.  ]
  [1.17 0.89 9.  ]]

 [[3.33 1.38 1.71]
  [2.6  2.   3.75]
  [3.2  8.5  2.25]]

 [[3.8  2.5  3.5 ]
  [3.14 5.75 6.  ]
  [5.   3.71  inf]]]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:21: RuntimeWarning: divide by zero encountered in true_div
```

Aula 09: Autovalores e autovetores

Autovalores

Definição

Seja A uma matriz quadrada de ordem n: $A_{n,n}$
Então um vetor não nulo $\vec{x}$ será autovetor de A se $A.\vec{x}$ for um múltiplo escalar de x:

$$A.\vec{x} = \lambda.\vec{x}$$

Obs: Autovetor é uma entidade que sempre aparece relacionada a um escalar (que chamamos de autovalor e aparece nas notações matemáticas como lambda λ).

Obs:
$|\lambda| > 1 \rightarrow$ Dilata o autovetor $\vec{x}$
$|\lambda| < 1 \rightarrow$ Retrai o autovetor $\vec{x}$
$|\lambda| = 0 \rightarrow$ Anula o autovetor $\vec{x}$
$|\lambda| < 0 \rightarrow$ Inverte o sentido do autovetor $\vec{x}$

Exemplo:
O vetor $X = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ é autovetor de $A = \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix}$, pois:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} . \begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix} = 3. \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Reescrevendo de acordo com a equação apresentada inicialmente:

$$\underset{A}{\begin{bmatrix} 3 & 0 \\ 8 & -1 \end{bmatrix}} \cdot \underset{\vec{x}}{\begin{bmatrix} 1 \\ 2 \end{bmatrix}} = \underset{\lambda}{3} \cdot \underset{\vec{x}}{\begin{bmatrix} 1 \\ 2 \end{bmatrix}}$$

No caso acima, dizemos que o escalar 3 (representado por λ) é o autovalor associado a A e $\vec{x}$ é o autovetor associado ao autovalor λ

Propriedades

Se $\vec{x}$ é autovetor associado a λ de A, então $k.\vec{x}$ também é autovetor de A, associado ao mesmo autovalor λ, onde k é um escalar.

Portanto, se λ é autovalor de A, então:
1- λ^k é autovalor de A^k
2- λ^{-1} é autovalor de A^{-1}
3- $k.\lambda$ é autovalor de $k.A$

Como encontrar o Autovalor?

Para encontrar o(s) autovalor(es), temos que resolver o sistema de equações proveniente da equação característica:

$$det(\lambda.\mathbb{I} - A) = 0$$

Exercícios de fixação

Exercício 2:

Encontrar os autovalores de $A = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$

Para encontrar os autovalores de A, recorremos à equação característica: det($\lambda.\mathbb{I} - A$) = 0

$$\begin{vmatrix} \lambda - 2 & -2 \\ -2 & \lambda - 2 \end{vmatrix} \iff (\lambda - 2)(\lambda - 2) - 4 = 0 \iff \lambda^2 - 4\lambda + 4 - 4 = 0 \iff \lambda^2 - 4\lambda = 0$$

Logo: $\lambda(\lambda - 4) = 0$ e temos que $\lambda = 0$ ou $\lambda = 4$

Resposta: Os autovalores são 0 ou 4

Exercício 3:

Encontrar os autovalores de $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 4 & 17 & 8 \end{bmatrix}$ e $B = \begin{bmatrix} 1/2 & 0 & 0 \\ -1 & 2/3 & 0 \\ 5 & -8 & 1 \end{bmatrix}$:

Basta aplicar a mesma lógica do exercício 2.

Observação: Da matriz B temos a seguinte propriedade:
Em matrizes triangulares (inferior ou superior), os autovalores (λ_1, λ_2,...) sempre correspondem aos valores da diagonal principal da matriz A

Autovetores

Serão melhores explorados na próxima aula

Aula 10: Autovalores, autovetores e autoespaços

Matriz Singular

Definição

É qualquer matriz que não admite inversa.

Propriedades

1- A é matriz singular se det(A) = 0
Macete: Linha ou coluna preenchida com elementos 0 significa, com certeza, que seu determinante é 0

2- Se houver pelo menos um autovalor $\lambda = 0$, então A é singular

3- Se existir um vetor $\vec{x}$ não nulo, tal que $A.\vec{x} = \vec{0}$

Traço

Definição

O traço é a soma dos elementos da diagonal principal de uma matriz:
tr(A_{nn}) $= a_{11} + a_{22} + \ldots a_{nn}$

Propriedades

1- tr(A) $=$ tr(A^t)
2- tr($A.B$) $=$ tr($B.A$)
3- tr(A) $= \lambda_1 + \lambda_2 + \lambda_3 + \ldots + \lambda_n$
4- det(A) $= \lambda_1 . \lambda_2 . \lambda_3 . \ldots . \lambda_n$

Afirmações equivalentes (Precisa saber)

1- λ é autovalor de A_{nn}
2- O sistema $(\lambda.\mathbb{I} - A).\vec{x} = \vec{0}$ de equações tem soluções não triviais
3- Existe um valor não nulo, tal que $A.\vec{x} = \vec{x}.\lambda$
4- λ é solução da equação característica $det(\lambda.\mathbb{I} - A) = 0$
Lembrando que a equação característica dá origem ao polinômio característico, cujas raízes são os autovalores λ associados à matriz A. A quantidade de raízes desse polinômio deve ser igual à dimensão da matriz.

Multiplicidade algébrica x Multiplicidade geométrica

Definição

Multiplicidade algébrica: É um número que se refere aos autovalores. Trata-se da quantidade de vezes que um mesmo autovalor λ é solução de um determinado polinômio característico.
Por exemplo, suponhamos que caímos no seguinte polinômio característico de 3º grau:

$$(\lambda).(\lambda).(\lambda - 3) = 0$$

Aqui, o 0 é raíz duas vezes e o 3 é raíz uma vez.
Dizemos, por isso, que o autovalor $\lambda = 0$ possui multiplicidade algébrica igual a 2 e o $\lambda = 3$ possui multiplicidade algébrica igual a 1.
Em notação, fica: ma(0) = 2 e ma(3) = 1

Multiplicidade geométrica: Número que se refere aos autovetores. Informa a quantidade de vetores na base do autoespaço que está associado a um determinado autovalor λ.
Notação: mg(λ)

Exercício de fixação

- **Seja:** $A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

Determine:

1- a multiplicidade algébrica da matriz A

2- a multiplicidade geométrica de cada autovetor

3- O autoespaço associado a cada autovalor de A

1º passo: Encontrar a multiplicidade algébrica

Para isso, vamos resolver a equação característica.

$$det(\lambda.\mathbb{I} - A) = 0 \iff \begin{vmatrix} \lambda & 0 & -1 \\ 0 & \lambda & -1 \\ 0 & 0 & \lambda - 1 \end{vmatrix} = 0 \iff (\lambda).(\lambda).(\lambda - 1) = 0$$

No polinômio característico encontrado temos que 0 é raiz duas vezes e 1 é raíz uma vez.

Portanto, o autovalor $\lambda = 0$ possui multiplicidade algébrica 2 e o autovalor $\lambda = 1$ possui multiplicidade algébrica 1. Em notação, dizemos que ma(0) = 2 e ma(1) = 1

Encontrada a multiplicidade algébrica, vamos ao 2º passo, que é determinar a multiplicidade geométrica associada a cada autovalor.

Como visto anteriormente, a multiplicidade geométrica vai depender do autovalor associado ao autovetor. Aqui temos os autovalores 0 e 1, por isso vamos começar com o autovalor 0:

Utilizamos: $(\lambda.\mathbb{I} - A).\vec{x} = \vec{0}$

Com $\lambda = 0$, temos: $(-A).\vec{x} = \vec{0}$

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & -1 \\ 0 & 0 & -1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \iff \begin{bmatrix} -x_3 \\ -x_3 \\ -x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Temos aqui que x_3 vale 0, portanto conseguimos escrever o vetor $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix}$ como a seguinte combinação linear:

$$\begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} = x_1 . \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + x_2 . \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Finalmente, como existem duas constantes (x_1 e x_2) que multiplicam os vetores unitários ($[1,0,0]$ e $[0,1,0]$, respectivamente) dizemos que para o autovalor $\lambda = 0$, a multiplicidade geométrica é igual a 2.

Lembrando que o pré-requisito para um vetor ser considerado autovetor, é ele ser não-nulo. Logo o vetor [0,0,0] encontrado na combinação linear não pertence ao autoespaço de $\lambda = 0$.

Também é possível dizer que o autoespaço associado ao autovalor 0 é composto pelos vetores [1,0,0] e [0,1,0]

Encontramos a multiplicidade geométrica para $\lambda = 0$, falta encontrar para o outro autovalor encontrado, que é $\lambda = 1$

Vamos repetir o procedimento:

$$(\lambda.\mathbb{I} - A).\vec{x} = \vec{0} \iff \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} x_1 - x_3 = 0 \\ x_2 - x_3 = 0 \end{cases}$$

Temos aqui infinitas soluções para o sistema, pois $x_1 = x_2 = x_3$, então podemos fazer infinitas combinações lineares de $\vec{x} = \begin{bmatrix} x_1 \\ x_1 \\ x_1 \end{bmatrix}$:

$$\vec{x} = x_1 . \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 \end{bmatrix}$$

Resposta final: As bases dos autoespaços associados a $\lambda = 0$ e $\lambda = 1$ são, respectivamente: {v1, v2} = {(1, 0, 0), (0, 1, 0)} e {v3} = {(1,1,1)}

Encontrando os autovetores

Para encontrar o autovetor (que é sempre associado a um determinado autovalor lambda, basta utilizar:

$$(\lambda . \mathbb{I} - A) . \vec{x} = \vec{0}$$

Obs:Essa equação irá resultar em um sistema, onde cada variável deve ser uma componente do vetor $\vec{x}$
Obs2: Exemplo prático em Aula 10> Multiplicidade algébrica x Multiplicidade geométrica>Exercício de fixação

Autoespaço

Definição

Autoespaço é o conjunto de todos os autovetores associados a um determinado autovalor união vetor nulo ($\vec{0}$)

Dito de outra forma, o autoespaço de A associado ao autovalor λ é o espaço solução do sistema homogêneo dos autovetores associados a um autovalor que garantem: $(\lambda . \mathbb{I} - A) . \vec{x} = \vec{0}$

Exercício de fixação

Exercício 01: Determinar λ para $A = \begin{bmatrix} 4 & 0 & 1 \\ 2 & 3 & 2 \\ 1 & 0 & 4 \end{bmatrix}$ e $\vec{x} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$

Para resolver essa questão, podemos utilizar tanto a equação característica $det(\lambda . \mathbb{I} - A) = 0$ ou então a equação $A . \vec{x} = \vec{x} . \lambda$
Aqui vamos utilizar arbitrariamente a segunda equação:

$$\begin{bmatrix} 4 & 0 & 1 \\ 2 & 3 & 2 \\ 1 & 0 & 4 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} . \lambda \iff \begin{bmatrix} 4.1 + 0.2 + 1.1 \\ 2.1 + 3.2 + 2.1 \\ 1.1 + 0.2 + 4.1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} . \lambda \iff \begin{bmatrix} 5 \\ 10 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} . \lambda$$

Portanto, temos que $\lambda = 5$

Obs: Nota-se que essa equação dá apenas um valor de lambda (não é possível inferir a multiplicidade algébrica deste autovalor), sendo que a matriz possui ordem n = 3. O método do polinômio característico é o que daria todos os três autovalores.

Exercício 02: Encontrar a equação característica para

$A = \begin{bmatrix} 10 & -9 \\ 4 & -2 \end{bmatrix}$ e para $B = \begin{bmatrix} 0 & 3 \\ 4 & 0 \end{bmatrix}$

Começamos com a matriz A:
$det(\lambda . \mathbb{I} - A) = 0$

$$\begin{vmatrix} \lambda - 10 & 9 \\ -4 & \lambda + 2 \end{vmatrix} = 0 \iff (\lambda - 10)(\lambda + 2) - (-4)(9) = 0 \iff \lambda^2 + 2\lambda - 10\lambda - 20 + 36 = 0 \iff \lambda^2 - 8\lambda + 16 =$$

Polinômio característico de A é $\lambda^2 - 8\lambda + 16 = 0$

Para a matriz B, temos o mesmo raciocínio:

$det(\lambda . \mathbb{I} - A) = 0$

$$\begin{vmatrix} \lambda & -3 \\ -4 & \lambda \end{vmatrix} = 0 \iff \lambda^2 - 12 = 0$$

Polinômio característico de B é $\lambda^2 - 12 = 0$

Aula 11: Diagonalização de Matrizes

Resolvendo sistemas lineares com matrizes

Um dos métodos possíveis (já apresentado neste documento) é calculando os determinantes de matrizes.

Outro método possível é através da fórmula

$$A . x = B$$

onde: A é a matriz de coeficientes, x a matriz de incógnitas da equação e B a matriz com os termos independentes.
Ilustrando:
O sistema:

$$\begin{cases} a_1 x_1 + b_1 x_2 + c_1 x_3 = d_1 \\ a_2 x_1 + b_2 x_2 + c_2 x_3 = d_2 \\ a_3 x_1 + b_3 x_2 + c_3 x_3 = d_3 \end{cases}$$

Pode ser resolvido através de $A . x = B$, sendo:

A = $\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}$; x= $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$; A = $\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$

Existe ainda uma terceira maneira de calcular um sistema linear através de matrizes. Para isso, vamos multiplicar a equação $A . x = B$ pela inversa de A em ambos os lados da igualdade:

$A . x = B \iff (A^{-1}) . A . x = B . (A^{-1}) \iff \mathbb{I} . x = B . A^{-1}$

Logo:

$$x = A^{-1} . B$$

E como eu calculo a inversa de uma matriz???

Existem alguns jeitos de descobrir a inversa de uma matriz. Um deles é através do seguinte método (**Método de Eliminação de Gauss-Jordan**):
Fazemos uma espécie de matriz aumentada onde colocamos em uma mesma matriz a própria matriz com os coeficientes do sistema e ao lado dela (dentro dessa nova matriz), a sua matriz identidade:

Usemos o seguinte sistema linear como exemplo:

$$\begin{cases} 2x - y = 2 \\ x - y = 1 \end{cases}$$

Vamos agora montar a "matriz aumentada" contendo o sistema e $\mathbb{I}$:

$$\left[\begin{array}{cc|cc} 2 & -1 & 1 & 0 \\ 1 & -1 & 0 & 1 \end{array}\right]$$

Agora, **o que queremos é transformar a parte que contém os coeficientes do sistema (que está à esquerda da barra) em uma matriz identidade, através das operações sobre linha de matriz***.
Na matriz acima, o primeiro passo é transformar o elemento a_{11} em 1. Para isso, fazemos:

$$L_1 \leftrightarrow L_1 - L_2 \implies \left[\begin{array}{cc|cc} 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \end{array}\right]$$

Agora, queremos zerar o elemento a_{21} (também através de operação de linhas):

$$L_2 \leftrightarrow L_2 - L_1 \implies \left[\begin{array}{cc|cc} 1 & 0 & 1 & -1 \\ 0 & -1 & -1 & 2 \end{array}\right]$$

Por último, queremos transformar o elemento a_{22} em 1:

$$L_2 \leftrightarrow (-1).L_2 \implies \left[\begin{array}{cc|cc} 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & -2 \end{array}\right]$$

Agora que transformamos a parte esquerda da matriz em uma identidade, a parte direita obtida (através dessas sucessivas operações sobre linhas) é a sua matriz inversa.

Com a matriz inversa em mãos, podemos voltar à equação $x = B.A^{-1}$ e encontramos o conjunto solução do sistema linear:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & -2 \end{bmatrix} . \begin{bmatrix} 2 \\ 1 \end{bmatrix} \iff \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2.1 + (-1).1 \\ 1.2 + (-2).1 \end{bmatrix} \iff \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Operações sobre as linhas de uma matriz

A tabela a seguir resume as três **operações elementares sobre linhas de uma matriz**.

Operação realizada na linha	Exemplo
Troque quaisquer duas linhas	$\begin{bmatrix} 2 & 5 & 3 \\ 3 & 4 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 4 & 6 \\ 2 & 5 & 3 \end{bmatrix}$ (Troca entre as linhas 1 e 2.)
Multiplique uma linha por uma constante diferente de zero	$\begin{bmatrix} 2 & 5 & 3 \\ 3 & 4 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \cdot 2 & 3 \cdot 5 & 3 \cdot 3 \\ 3 & 4 & 6 \end{bmatrix}$ (A linha 1 é multiplicada por 3.)
Some duas linhas	$\begin{bmatrix} 2 & 5 & 3 \\ 3 & 4 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 5 & 3 \\ 3+2 & 4+5 & 6+3 \end{bmatrix}$ (A linha 2 se torna a soma das linhas 2 e 1.)

As operações sobre linhas de uma matriz podem ser usadas para resolver sistemas de equações, mas antes de analisarmos o porquê, vamos praticar um pouco.

*Mais informações em https://pt.khanacademy.org/math/precalculus/precalc-matrices/elementary-matrix-row-operations/a/matrix-

Diagonalização de Matrizes

Definição:

Diagonalização é o processo de encontrar uma matriz diagonal correspondente a uma matriz ou operador diagonalizável. É utilizada devido ao fato que compartilha uma série de propriedades com a matriz diagonalizada e tem um custo de processamento muito inferior.
É portanto um procedimento muito utilizado para processamento de matrizes de ordem muito grandes

Propriedades:

A matriz diagonal compartilha das seguintes características com a matriz original:

- As características (postos) são iguais
  obs: posto é a quantidade de linhas não nulas de uma matriz
- As nulidades são iguais
  obs: nulidade = n - p, onde n é o número de colunas da matriz e p é o posto
- Os determinantes são iguais
- Os traços são iguais
- Os autovalores λ são iguais
- Os autovetores são correspondentes

Condição/pré-requisito para diagonalização:

1. Uma matriz é diagonalizável se e somente se:
   $$B = P^{-1}.A.P$$
   Onde: B é a matriz diagonal, A é a matriz diagonalizada e P é a matriz que diagonaliza A
2. A é diagonalizável se, e somente se, a multiplicidade geométrica de cada autovetor for igual à multiplicidade algébrica do autovalor associado

Como diagonalizar?

Matriz diagonal (B)

Para determinar a matriz diagonal, partimos da propriedade de que a matriz diagonal compartilha os mesmos autovalores da matriz original, logo:

B = $\begin{bmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda_n \end{bmatrix}$, sendo $\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_n$ autovalores de A

onde B é a matriz diagonal equivalente a A

Matriz P (que diagonaliza A)

É a matriz composta pelos autovetores de A (desde que os critérios acima pontuados sejam preenchidos)

Exercício de fixação

Encontre a matriz que diagonaliza $A = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

Resolução:

A matriz que diagonaliza A é a matriz P (passo 5). Para fins didáticos, vamos encontrar também P^{-1}

Passos:

1. Determinar autovalores de A
2. Encontrar autovetores associados aos autovalores de A
3. Certificar que ma(λ_1) >= mg(λ_1), ma(λ_2) >= mg(λ_2), etc.
4. Certificar que os autovetores obtidos são linearmente independentes
5. Obter P através dos autovetores encontrados
6. Obter P^{-1} através de método gaussiano

Passo 1-

$$det(\lambda.\mathbb{I} - A) = 0 \iff \begin{vmatrix} \lambda - 2 & 0 & -1 \\ -1 & \lambda & -1 \\ 0 & 0 & \lambda - 1 \end{vmatrix} = 0 \iff (\lambda - 2)(\lambda)(\lambda - 1) = 0$$

Logo:

$\lambda_1 = 2 \rightarrow$ ma(2) = 1

$\lambda_2 = 0 \rightarrow$ ma(0) = 1

$\lambda_3 = 1 \rightarrow$ ma(1) = 1

Passo 2-

2.1- Para $\lambda_1 = 2$

$$(\lambda.\mathbb{I} - A).\vec{x} = \vec{0} \iff \begin{bmatrix} \lambda - 2 & 0 & -1 \\ -1 & \lambda & -1 \\ 0 & 0 & \lambda - 1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \iff$$

$$\begin{bmatrix} 0 & 0 & -1 \\ -1 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} -x_3 = 0 \\ -x_1 + 2.x_2 - x_3 = 0 \\ x_3 = 0 \end{cases}$$

$x_3 = 0, x_1 = -2x_2$

Temos, portanto, que para $\lambda = 2$:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2x_2 \\ x_2 \\ 0 \end{bmatrix} = x_2 \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}$$

2.2 - Para $\lambda_2 = 0$

$$(\lambda.\mathbb{I} - A).\vec{x} = \vec{0} \iff \begin{bmatrix} \lambda - 2 & 0 & -1 \\ -1 & \lambda & -1 \\ 0 & 0 & \lambda - 1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \iff$$

$$\begin{bmatrix} -2 & 0 & -1 \\ -1 & 0 & -1 \\ 0 & 0 & -1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} -2x_1 - x_3 = 0 \\ -x_1 - x_3 = 0 \\ -x_3 = 0 \end{cases}$$

$x_1 = x_3 = 0$

Temos, portanto, que para $\lambda = 0$:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ x_2 \\ 0 \end{bmatrix} = x_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

2.3 - Para $\lambda_3 = 1$

$$(\lambda.\mathbb{I} - A).\vec{x} = \vec{0} \iff \begin{bmatrix} \lambda - 2 & 0 & -1 \\ -1 & \lambda & -1 \\ 0 & 0 & \lambda - 1 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \iff$$

$$\begin{bmatrix} -1 & 0 & -1 \\ -1 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{cases} -x_1 - x_3 = 0 \\ -x_1 + x_2 - x_3 = 0 \end{cases}$$

$x_1 = x_3, x2 = x1 + x3$

Temos, portanto, que para $\lambda = 1$:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ 2x_1 \\ x1 \end{bmatrix} = x_1 \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

Os autovetores associados aos seus respectivos autovalores ($\lambda_1 = 2$,$\lambda_2 = 0$ e $\lambda_3 = 1$) são $[-2, 1, 0]$, $[0, 1, 0]$ e $[1, 2, 1]$

Passo 3-

ma(2) <= mg(2)

ma(0) <= mg(0)

ma(3) <= mg(3)

Passo 4-

Calcula-se o determinante da matriz formada pelos autovetores obtidos (também chamada P)

$$\begin{vmatrix} -2 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{vmatrix} = -2$$

Determinante $\neq$ 0, logo os autovetores obtidos são L.I.

Passo 5-

P = $\begin{bmatrix} -2 & 0 & 1 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$

Passo 6-

Obtendo P^{-1} (método da matriz aumentada):

$$\left[\begin{array}{ccc|ccc} -2 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow L_1 \leftrightarrow L_1 + 3.L_2 \backslash$$

$$= \left[\begin{array}{ccc|ccc} 1 & 3 & 7 & 1 & 3 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow L_2 \leftrightarrow L_2 - L_1 \backslash$$

$$= \left[\begin{array}{ccc|ccc} 1 & 3 & 7 & 1 & 3 & 0 \\ 0 & -2 & -5 & -1 & -2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow L_2 \leftrightarrow (\frac{-1}{2}). L_2 \backslash$$

$$= \left[\begin{array}{ccc|ccc} 1 & 3 & 7 & 1 & 3 & 0 \\ 0 & 1 & \frac{5}{2} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow L_1 \leftrightarrow L_1 - 3. L_2 \backslash$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & \frac{-1}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & 1 & \frac{5}{2} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow L_1 \leftrightarrow L_1 + \frac{1}{2}L_3 \text{ e } L_2 \leftrightarrow L_2 + \frac{-5}{2}L_3 \backslash$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{-1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 & \frac{1}{2} & 1 & \frac{-5}{2} \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Portanto, temos $P^{-1} = \begin{bmatrix} \frac{-1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{-5}{2} \\ 0 & 0 & 1 \end{bmatrix}$

Observação importante: Para os outros exercícios, não basta apenas encontrarmos as matrizes P e P^{-1}, é preciso checar se a condição $B = P^{-1}. A. P$ é verdadeira, para assumirmos que existe matriz diagonal para A

Aula 12: Decomposição (ou fatoração) de matrizes LU

Definição

A decomposição (ou fatoração) de matrizes L.U. facilita a resolução de sistemas com muitas variáveis.
A fatoração acontece de modo a transformar uma matriz *A*, de modo que:

$$A = L.U$$

Onde A é a matriz formada pelos coeficientes da equação e L e U, respectivamente, correspondem a lower e upper (matrizes triangular inferior e superior) da matriz A modificada (isso significa que através da matriz original A, através de operações elementares, podemos obter as matrizes L e U)

Se recuperarmos a fórmula já vista de resolução de sistemas lineares com matrizes, $A.\vec{x} = B$, temos:

$$\begin{cases} A.\vec{x} = B \\ A = L.U \end{cases} \iff L.U.\vec{x} = B$$

Ao acrescentarmos $U.\vec{x} = \vec{y}$, seguimos com:
$L.\vec{y} = B$

Portanto, para encontrarmos o vetor solução ($\vec{x}$), precisamos:

1. Determinar L e U de A;
2. Resolver $L.\vec{y} = B$, de modo a obter $\vec{y}$;
3. Resolver $U.\vec{x} = \vec{y}$, de modo a obter o vetor solução do sistema ($\vec{x}$)

Obs: É possível que existam pivôs nulos. Isso significa que algumas matrizes podem ter os pivôs (ou seja, os elementos da diagonal principal) iguais a 0, não podendo ser usados para zerar outras linhas da matriz A, e consequentemente tornando impossível a determinação da matriz triangular superior (U). Para esses casos, utilizamos a técnica de pivotagem parcial (aula 13)

Mas quem diabos é L e U???

Lower (L): É a matriz triangular inferior, com elementos da diagonal principal iguais a 1 e os demais elementos iguais aos multiplicadores das etapas de escalonamento da matriz superior (U):

$$L = \begin{bmatrix} 1 & 0 & 0 \\ mL_{21} & 1 & 0 \\ mL_{31} & mL_{32} & 1 \end{bmatrix}$$

mL_{21}, mL_{31} e mL_{32} são chamados de fatores de A e podem ser obtidos através da divisão do elemento da posição correspondente pelo pivô da coluna ($\frac{elemento}{pivô}$)

Por exemplo, no exemplo acima temos:
$mL_{21} = \frac{A_{21}}{pivô}, mL_{31} = \frac{A_{31}}{pivô}$ e $mL_{32} = \frac{A_{32}}{pivô}$
Lembrando que o pivô é o mesmo para elementos de mesma coluna

Upper (U): Matriz triangular superior, resultante do escalonamento de A. É o resultado da matriz A após a obtenção da matriz L
Portanto, ambas as matrizes são obtidas através do escalonamento de A (mais informações do passo a passo, consultar este vídeo)
Dica: Começar escalonando a matriz A (a fim de se obter U), e os fatores de A (que correspondem os elementos não nulos de L) vão sendo descobertos

Requisitos

Para se decompôr uma matriz A através do método LU...

1. Precisa ser quadrada (A_{mn}, onde m = n)
2. Não pode ser singular (ou seja, $det(A) \neq 0$)

Exercício de fixação

Resolver o sistema por decomposição LU:

$$\begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix}$$

Passo a passo da resolução:

1. Encontrar L e U (método de Gauss)
2. Calcular $L.\vec{y} = B$ (lembrando que y = U.x)
3. Resolver $\vec{y} = U.\vec{x}$
4. Vetor $\vec{x}$ é o vetor-solução do sistema linear 🤡

Passo 1 (esse passo é ctrl c ctrl v desse vídeo):

- Encontrando U: Precisamos transformar a matriz A em uma triangular superior, por isso zeramos a primeira coluna (os elementos abaixo do elemento da diagonal principal) utilizando o elemento a_{11} como pivô:

$$A_0 = \begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix}$$

pivô:a_{11}, $mL_{21} = -2$ e $L_2 \leftarrow L_2 - mL2.L_1 \iff A_1 = \begin{bmatrix} 1 & -3 & 2 \\ 0 & 2 & 3 \\ 4 & -6 & 5 \end{bmatrix}$

pivô:a_{11}, $mL_{31} = 4$ e $L_3 \leftarrow L_3 - mL3.L_1 \iff A_2 = \begin{bmatrix} 1 & -3 & 2 \\ 0 & 2 & 3 \\ 0 & 6 & -3 \end{bmatrix}$

Zerados os elementos da primeira coluna abaixo da diagonal principal (a_{21} e a_{31}), precisamos zerar o elemento abaixo da diagonal principal na segunda coluna (a_{32}) para concluirmos a transformação em uma matriz triangular superior. O pivô, então passa a ser o elemento a_{22}:

pivô:a_{22}, $mL_{32} = 3$ e $L_3 \leftarrow L_3 - mL3.L_1 \iff A_3 = \begin{bmatrix} 1 & -3 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & -12 \end{bmatrix} = U$

- Encontrando L:

Como já visto, em uma decomposição LU, a matriz L pode ser expressa do seguinte modo: $L = \begin{bmatrix} 1 & 0 & 0 \\ mL_{21} & 1 & 0 \\ mL_{31} & mL_{32} & 1 \end{bmatrix}$

Sendo que mL_{21}, mL_{31} e mL_{32} foram obtidos durante o processo de determinação de U. Logo:

Com $mL_{21} = -2, mL_{31} = 4$ e $mL_{32} = 3$, temos:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix}$$

Passo 2: Com L e U em mãos, precisamos resolver a equação $L.\vec{y} = B$

Assumindo $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$, temos:

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix}$$

$$\begin{cases} y_1 = 11 \\ -2y_1 + y_2 = -15 \\ 4y_1 + 3y_2 + y_3 = 29 \end{cases}$$

Portanto, temos $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 7 \\ -36 \end{bmatrix}$

Passo 3: com $\vec{y}$ em mãos, basta resolver $\vec{y} = U.\vec{x}$

$$\begin{bmatrix} 11 \\ 7 \\ -36 \end{bmatrix} = \begin{bmatrix} 1 & -3 & 2 \\ 0 & 2 & 3 \\ 0 & 0 & -12 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{cases} x - 3y + 2z = 11 \\ 2y + 3z = 7 \\ -12z = -36 \end{cases}$$

Logo, temos $\vec{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$

Passo 4: Vetor $\vec{x}$ representa o conjunto solução do sistema linear

Logo: $x = 2, y = -1$ e $z = 3$

Fatoração LU em Python

Algoritmo construído que retorna L e U dado uma matriz A

```
import numpy as np
def fatoraLU(A):
    U = np.copy(A)
    n = np.shape(U)[0]
    L = np.eye(n)
    for j in np.arange(n-1):
        for i in np.arange(j+1,n):
            L[i,j] = U[i,j]/U[j,j]
            for k in np.arange(j+1,n):
                U[i,k] = U[i,k] - L[i,j]*U[j,k]
            U[i,j] = 0
    return L, U
```

Aula 13: Decomposição (ou fatoração) de matrizes

Pivotagem parcial

Definição

É um método complementar à decomposição de matriz LU. Serve nos casos onde pelo menos um dos pivôs da matriz A é nulo. Como sabemos, um pivô não pode ser nulo, pois caso seja, ele não pode zerar outras linhas da matriz (impedindo a determinação de L e U).

Nesse caso, utilizamos uma matriz auxiliar P, também chamada de matriz de permutação, (que nada mais é do que uma matriz identidade $\mathbb{I}$ com linhas trocadas). As linhas trocadas devem ser aquelas de A de modo que o pivô nulo seja trocado por um não nulo (e portanto possa ser possível calcular as matrizes L e U).

Aproveitando a equação $A.\vec{x} = B$, multiplicamos por P em ambos os lados, ficando:

$$P.A.\vec{x} = P.B$$

Após a inversão das linhas de A (com o auxílio de P), a resolução do sistema linear segue idêntica ao método de decomposição LU.
Aqui vale a observação de que na equação apresentada acima $P.A.\vec{x} = P.B$ a matriz P irá inverter linhas de A e irá inverter colunas de B.
Isso é importante pois a ordem de P no produto por uma matriz pode alterar a linha ou a coluna, dependendo de sua posição no produto ($A.P \neq P.A$)

Curiosidade 1 - Demonstração que o produto de P por A inverte as linhas de A:

Seja $A = \begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix}$ e $P_{13} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ (onde P_{13} indica que trocamos as linhas 1 e 3 da matriz identidade de dimensão 3)

$$P.A = \begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} . \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & -6 & 5 \\ -2 & 8 & -1 \\ 1 & -3 & 2 \end{bmatrix}$$

Curiosidade 2 - P à esquerda no produto inverte as linhas e P à direita no produto inverte as colunas

$P_{13}.A \rightarrow A$ com as linhas 1 e 3 trocadas

$A.P_{13} \rightarrow A$ com as colunas 1 e 3 trocadas

```
import numpy as np
A = np.array([[1, -3, 2],
              [-2, 8, -1],
              [4, -6, 5]])
P = np.array([[0, 0, 1],
              [0, 1, 0],
              [1, 0, 0]]) #matriz de permutação das linhas 1 e 3
print("""A:
{}
""".format(A))
print("""P*A inverte linhas:
{}
""".format(P.dot(A)))
print("""A*P inverte colunas:
{}
""".format(A.dot(P)))
```

```
A:
[[ 1 -3  2]
 [-2  8 -1]
 [ 4 -6  5]]

P*A inverte linhas:
[[ 4 -6  5]
 [-2  8 -1]
 [ 1 -3  2]]

A*P inverte colunas:
[[ 2 -3  1]
 [-1  8 -2]
 [ 5 -6  4]]
```

Exemplo

$$\begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} . \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix}$$

Ps: Este problema é o mesmo resolvido na aula passada por decomposição LU e como não há a necessidade de usarmos pivotagem parcial já que não há pivôs nulos nesta matriz, vamos resolver utilizando esse recurso e verificarmos que o conjunto solução permanece o mesmo:

Ps2: Aqui vamos utilizar a matriz $P_{12} = (2, 1, 3)$, ou seja, vamos inverter as linhas 1 e 2 da matriz A

Resolução (passo a passo)

1. Resolver $P_{12}.A$ e $P_{12}.B$, da equação $P_{12}.A.\vec{x} = P_{12}.B$
2. Determinar L e U
3. Resolver $L.\vec{y} = P_{12}.B$
4. Resolver $U.\vec{x} = \vec{y}$ para encontrar o vetor solução $\vec{x}$

Passo 1:

$$P_{12}.A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & -3 & 2 \\ -2 & 8 & -1 \\ 4 & -6 & 5 \end{bmatrix} = \begin{bmatrix} -2 & 8 & -1 \\ 1 & -3 & 2 \\ 4 & -6 & 5 \end{bmatrix}$$

$$P_{12}.B = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 11 \\ -15 \\ 29 \end{bmatrix} = \begin{bmatrix} -15 \\ 11 \\ 29 \end{bmatrix}$$

Como esperado, após o produto por P_{12}, as linhas 1 e 2 das matrizes A e B foram trocadas.

Passo 2:

$$A_0 = \begin{bmatrix} -2 & 8 & -1 \\ 1 & -3 & 2 \\ 4 & -6 & 5 \end{bmatrix}$$

pivô: a_{11}

$mL_{21} = \frac{1}{-2} = -0.5$

$mL_{31} = \frac{4}{-2} = -2$

$$A_1 = \begin{bmatrix} -2 & 8 & -1 \\ 0 & 1 & 1.5 \\ 0 & 10 & 3 \end{bmatrix}$$

pivô: a_{22}

$mL_{32} = \frac{10}{1} = 10$

$$A_2 = \begin{bmatrix} -2 & 8 & -1 \\ 0 & 1 & 1.5 \\ 0 & 0 & -12 \end{bmatrix} = U$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ mL_{21} & 1 & 0 \\ mL_{31} & mL_{32} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ -2 & 10 & 1 \end{bmatrix}$$

Passo 3:

$$L.\vec{y} = P_{12}.B \iff \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ -2 & 10 & 1 \end{bmatrix} . \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -15 \\ 11 \\ 29 \end{bmatrix}$$

$$\begin{cases} y_1 = -15 \\ -0.5y_1 + y_2 = 11 \\ -2y_1 + 10y_2 + y_3 = 29 \end{cases}$$

$y_1 = -15, y_2 = 3.5, y_3 = -36$

Passo 4:

$$U.\vec{x} = \vec{y} \iff \begin{bmatrix} -2 & 8 & -1 \\ 0 & 1 & 1.5 \\ 0 & 0 & -12 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -15 \\ 3.5 \\ -36 \end{bmatrix}$$

$$\begin{cases} -2x + 8y - z = -15 \\ y + 1.5z = 3.5 \\ -12z = -36 \end{cases}$$

$x = 2, y = -1, z = 3$

Portanto, como visto na aula passada e como esperado, a solução permanece a mesma, ou seja: $\vec{x} = \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}$

Pivotagem parcial em Python

```
#A Decomposição LU pode ser realizada usando a função lu() do módulo linalg
#da biblioteca scipy
#Aqui é usada a pivotagem parcial: PA = LU ou A = LUP
#Logo as matrizes resultantes não serão as mesmas.

import numpy as np
from scipy.linalg import lu

A = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
print("Matriz A: \n",A)
det = np.linalg.det(A)
print("\nDeterminante de A: ",det)
```

```
P, L, U = lu(A)
print("\nP: \n",P)
print("\nL: \n",L)
print("\nU: \n",U)
```

```
Matriz A:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Determinante de A:  0.0

P:
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]

L:
 [[1.   0.   0.  ]
 [0.14 1.   0.  ]
 [0.57 0.5  1.  ]]

U:
 [[7.   8.   9.  ]
 [0.   0.86 1.71]
 [0.   0.   0.  ]]
```

No exemplo acima, como o determinante da matriz $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ é nulo (ou seja, não é uma matriz inversível), foi necessário utilizar a pivotagem parcial para inverter as linhas e torná-la uma matriz inversível.

Sistemas lineares em Python

O método solve, do módulo linalg do Numpy, é uma implementação da fatoração LU vista acima
https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html

```
import numpy as np
A = np.array([[1, -3, 2],
              [-2, 8, -1],
              [4, -6, 5]])
b = np.array([11, -15, 29])
x = np.linalg.solve(A, b)
print(x)
```

```
[ 2. -1.  3.]
```

Decomposição de Cholesky

Definição

Assim como na decomposição LU, a decomposição de Cholesky visa decompor uma matriz A (que representa os coeficientes de um sistema linear) de modo a facilitar resolução de um sistema linear.
A diferença reside em, ao invés de decompormos A em $L.U$ (como é feito na decomposição LU), decompomos A em $L.L^t$.
Importante lembrar que a matriz L da fatoração LU não é a mesma matriz L da decomposição de Cholesky.

O restante da resolução, que parte de $A.\vec{x} = B$, é muito semelhante ao procedimento já visto anteriormente:

1. Determinar L e L^t
2. Calcular $\vec{y}$ em $L.\vec{y} = B$ (lembrando que $\vec{y} = L^t.\vec{x}$)
3. Com $\vec{y}$ determinado, calcular o vetor solução $\vec{x}$ através de $\vec{y} = L^t.\vec{x}$

Vantagem da decomposição de Cholesky

Quando os requisitos são atendidos, a fatoração de Cholesky é mais rápida do que a fatoração LU

Requisitos

1. A deve ser uma matriz simétrica, ou seja, $A = A^t$
2. A deve ser uma matriz definida positiva, ou seja, $\overrightarrow{v^t} . A . \vec{v} > 0, \forall \vec{v} \neq \vec{0}$

Exercício

Encontrar a matriz L que fatora $A = \begin{bmatrix} 4 & -2 & 2 \\ -2 & 10 & -7 \\ 2 & -7 & 30 \end{bmatrix}$

Solução manual

Partimos de $L . L^t = A$

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} . \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix} = \begin{bmatrix} 4 & -2 & 2 \\ -2 & 10 & -7 \\ 2 & -7 & 30 \end{bmatrix} \iff$$

$$\begin{bmatrix} (l_{11})^2 & l_{11} . l_{21} & l_{11} . l_{31} \\ l_{21} . l_{11} & (l_{21})^2 + (l_{22})^2 & l_{21} . l_{31} + l_{22} . l_{32} \\ l_{31} . l_{11} & l_{31} . l_{21} + l_{32} . l_{22} & (l_{31})^2 + (l_{32})^2 + (l_{33})^2 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 2 \\ -2 & 10 & -7 \\ 2 & -7 & 30 \end{bmatrix}$$

Resolvendo a primeira linha:

$(l_{11})^2 = 4 \iff l_{11} = 2$

$l_{11} . l_{21} = -2 \iff l_{21} = \frac{-2}{2} = -1$

$l_{11} . l_{31} = 2 \iff l_{31} = \frac{2}{2} = 1$

Resolvendo a segunda linha:

$(l_{21})^2 + (l_{22})^2 = 10 \iff l_{22} = \sqrt{10 - 1} = 3$

$l_{21} . l_{31} + l_{22} . l_{32} = -7 \iff -1.1 + 3 . l_{32} = -7 \iff l_{32} = -2$

Resolvendo a terceira linha:

$(l_{31})^2 + (l_{32})^2 + (l_{33})^2 = 30 \iff 1 + 4 + (l_{33})^2 = 30 \iff (l_{33})^2 = 25 \iff l_{33} = 5$

Temos como L, portanto:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 3 & 0 \\ 1 & -2 & 5 \end{bmatrix}$$

Solução no Python

```
import numpy as np
A = np.array([[4, -2, 2],
              [-2, 10, -7],
              [2, -7, 30]])
L = np.linalg.cholesky(A)
print("""
A:
{}
""".format(A))
print("""
L:
{}
""".format(L))

# Como em Cholesky, decompomos A como L * L.t, vamos verificar
# retroativamente se L * L.t resulta em A:
print("""
L * L.T:
{}
""".format(L.dot(L.T)))
```

```
A:
[[ 4 -2  2]
```

```
 [-2 10 -7]
 [ 2 -7 30]]


L:
[[ 2.  0.  0.]
 [-1.  3.  0.]
 [ 1. -2.  5.]]


L * L.T:
[[ 4. -2.  2.]
 [-2. 10. -7.]
 [ 2. -7. 30.]]
```

Decomposição QR

Definição

Trata-se de um método de decomposição de matrizes não quadradas (A_{MxN}), tal que:

$$A = Q.R$$

Q: matriz quadrada (dimensão m), cujas colunas formam uma **base ortonormal**, ou seja, além de ser uma base ortogonal, seus vetores são unitários. $Q^T.Q = \mathbb{I}$

Lembrando que ser **ortogonal** significa que o **produto interno** entre pares de vetores distintos dessa base são igual a **zero**.

Além disso, uma **matriz ortogonal** é sempre invertível e sua inversa é dada pela sua transposta.

R: mxn, triangular superior, **não singular** (admite inversa), já que sua diagonal principal contém apenas termos não-nulos.

Requisitos

1. Vetores-coluna da matriz A devem ser Linearmente Independentes (LI)

Exemplo

```
import numpy as np
A = np.array([[1, 2],
              [3, 4],
              [5, 6]])
print("A: \n",A)

Q, R = np.linalg.qr(A)
print("\nQ: \n",Q)
print("\nR: \n",R)

#Reconstrução
B = Q.dot(R)
#B == A
print("\nMatriz A reconstruída: \n",B)

print("\nVerificando que Q forma uma base ortonormal: \n")
C = Q.T.dot(Q)
print(C)
```

```
A:
 [[1 2]
 [3 4]
 [5 6]]

Q:
 [[-0.17  0.9 ]
 [-0.51  0.28]
 [-0.85 -0.35]]

R:
 [[-5.92 -7.44]
 [ 0.    0.83]]

Matriz A reconstruída:
 [[1. 2.]
 [3. 4.]
 [5. 6.]]
```

```
Verificando que Q forma uma base ortonormal:

[[1. 0.]
 [0. 1.]]
```

Aula 14: Métodos iterativos para sistemas lineares

Método iterativo Gauss-Jacobi

Definição

Trata-se de um método que visa solucionar um sistema linear $A.\vec{x} = B$, de dimensões $n_x n$

Partindo-se de um vetor genericamente representado por $\overrightarrow{x^0} = (x_1^0, x_2^0, x_3^0, x_4^0)^T$, como $\overrightarrow{x^0} = (0,0,0,0)^T$, utiliza-se a técnica de Gauss-Jacobi para determinar k aproximações para $\vec{x}$ até que a seguinte condição seja feita:

$$\frac{||x^k - x^{k-1}||\infty}{||x^k||\infty} < 10^{-3}$$

Essa condição é chamada de **erro** e indica o mínimo de casas decimais de imprecisão para a aproximação ser aceita. No caso ilustrado acima, só serão aceitas aproximações, geradas pelo método, com erros à partir da 4ª casa decimal.

Vale notar que k refere-se às iterações e as normas são calculadas de acordo com o valor máximo. Por exemplo na quinta iteração (k=5), temos o erro expresso por:

$$k = 5 \rightarrow \frac{||x^5 - x^4||\infty}{||x^5||\infty} < 10^{-3} \iff \frac{max\{||x_1^5 - x_1^4||, ||x_2^5 - x_2^4||, ||x_3^5 - x_3^4||, ||x_4^5 - x_4^4||\}}{||max\{x_1^5, x_2^5, x_3^5, x_4^5\}||} < 10^{-3}$$

Exemplo

Resolva o seguinte sistema através do método iterativo Gauss-Jacobi: $\begin{cases} 10x_1 - x_2 + 2x_3 = 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 = 25 \\ 2x_1 - x_2 + 10x_3 - x_4 = -11 \\ 3x_2 - x_3 + 3x_4 + 8x_4 = 15 \end{cases}$

Resolução

Para resolver o sistema através do método iterativo de Gauss-Jacobi, vamos utilizar sucessivas k aproximações de $\overrightarrow{x^k} = (x_1^k, x_2^k, x_3^k, x_4^k)^T$, partindo de k = 0 como um vetor nulo, ou seja: $\overrightarrow{x^0} = (0,0,0,0)^T$

Agora, o que devemos fazer é isolar as variáveis do sistema e calcular que valores que elas (x_1, x_2, x_3, x_4) passam a assumir com a iteração realizada e comparar com o critério de parada. Caso o critério não seja atingido, uma nova iteração é iniciada, sendo que os novos valores de $\vec{x}$ passam a ser os últimos obtidos x^k (e na hora de comparar novamente com o critério, os valores antigos são x^{k-1})

Passo 1: Isolar as variáveis do sistema $\begin{cases} x_1 = \frac{x_2}{10} + \frac{-x_3}{3} + \frac{3}{5} \\ x_2 = \frac{x_1}{11} + \frac{x_3}{11} - \frac{3x_4}{4} + \frac{25}{11} \\ x_3 = \frac{-x_1}{5} + \frac{x_2}{10} + \frac{x_4}{10} - \frac{11}{10} \\ x_4 = \frac{-3x_2}{8} + \frac{x_3}{8} + \frac{15}{8} \end{cases}$

Passo 2: Iteração

Agora substituímos nosso vetor inicial (k=0) $\overrightarrow{x^0} = [x_1^0, x_2^0, x_3^0, x_4^0]^t = [0,0,0,0]^t$ no sistema com as variáveis já isoladas e obtemos:

$x_1 = \frac{x_2}{10} + \frac{-x_3}{3} + \frac{3}{5} = \frac{3}{5} = 0.6$

$x_2 = \frac{x_1}{11} + \frac{x_3}{11} - \frac{3x_4}{4} + \frac{25}{11} = \frac{25}{11} = 2.2727$

$x_3 = \frac{-x_1}{5} + \frac{x_2}{10} + \frac{x_4}{10} - \frac{11}{10} = -\frac{11}{10} = -1.1$

$x_4 = \frac{-3x_2}{8} + \frac{x_3}{8} + \frac{15}{8} = \frac{15}{8} = 1.875$

Ou seja, a próxima iteração (k=1) será feita com $\overrightarrow{x^1} = [x_1^1, x_2^1, x_3^1, x_4^1]^t = [0.6, 2.2727, -1.1, 1.875]^t$

Mas antes de partir para a próxima iteração, precisamos calcular o erro para avaliar se já é aceitável:

Para k = 1, o erro é: $\frac{||x^k - x^{k-1}||\infty}{||x^k||\infty} = \frac{max\{||x_1^1 - x_1^0||, ||x_2^1 - x_2^0||, ||x_3^1 - x_3^0||, ||x_4^1 - x_4^0||\}}{max\{||x_1^1||, ||x_2^1||, ||x_3^1||, ||x_4^1||\}} = \frac{max\{0.6, 2.2727, 1.1, 1.875\}}{max\{0.6, 2.2727, 1.1, 1.875\}} = \frac{2.2727}{2.2727} = 1$

Como o resultado do erro obtido é muito maior que 10^{-3}, que é a tolerância mínima de erro, então seguimos para a próxima iteração (k=2) e repetimos todo esse processo, ou seja, substituímos o vetor de k=1 (ou seja, $[0.6, 2.2727, -1.1, 1.875]^t$) no sistema linear com as variáveis isolados e obtemos o novo vetor ($\overrightarrow{x^2}$) para k=2, e repetimos a checagem do erro obtido e vemos se ele preenche a condição, aqui $> 10^3$.

Como a solução desse problema manualmente seria muito mais extensa e a solução só seria atingida na 10ª iteração (k=10), então o exemplo acima ilustra como funciona a primeira iteração do método. A solução é melhor implementada em um algoritmo.

De qualquer maneira, a cada iteração o vetor $\vec{x}$ obtido vai se aproximando (convergindo) para a solução real do sistema, que nesse

Método iterativo Gauss-Seidel

Definição

Segue o mesmo princípio do método iterativo Gauss-Jacobi, ou seja, também serve para encontrar a solução de um sistema linear através de aproximações sucessivas (k).

A diferença é que este método tende a ser mais eficaz pois "aproveita" os valores da iteração corrente (diferentemente do método de Gauss-Jacobi, que sempre compara a iteração corrente com a anterior).

Exemplo:

$$\begin{cases} x_1^k = ax_2^{k-1} + bx_3^{k-1} + cx_4^{k-1} + d \\ x_2^k = ex_1^k + fx_3^{k-1} + gx_4^{k-1} + h \\ x_3^k = ix_1^k + jx_2^k + lx_4^{k-1} + m \\ x_4^k = nx_1^k + ox_2^k + px_3^k + h \end{cases}$$

Na ilustração acima, podemos ver que embora x_1^k ainda utilize todos os valores obtidos a partir da iterçaão $k-1$, as variáveis seguintes começam a calcular seus valores progressivamente de acordo com a iteração atual:

A variável x_2^k utiliza os valores correspondentes da iteração k para a variável x_1 (embora permaneça com os valores correspondentes a k-1 para x_3 e x_4).

Enquanto a variável anterior usava apenas um valor da iteração k no seu cálculo, x_3^k utiliza os valores correspondentes à iteração k para as variáveis x_1 e x_2 (permanecendo com o valor da iteração anterior k-1 para x_4).

Por último, a variável x_4^k utiliza todos os valores da iteração corrente para x_1, x_2 e x_3.

Vemos então que há um ganho de performance do método já que em uma mesma iteração, há o cálculo das variáveis baseadas **também** em valores da iteração corrente (k), e não só da iteração anterior (k-1) como no método iterativo Gauss-Jacobi

Exemplo

O exemplo do método anterior que foi resolvido em 10 iterações, seria resolvido em 5 iterações com o método Gauss-Seidel

Aula 15: Autodecomposição de Matrizes

Consultar material da aula no Onedrive

Aula 16: Decomposição em Valores Singulares

Consultar material da aula no Onedrive

✓ 0s conclusão: 22:09