pcv

April 22, 2022

[1]:
```
import random
from itertools import permutations
import matplotlib.pyplot as plt
```

[2]:
```
def distancia(ponto1, ponto2):
    """
    Função que calcula e retorna a distância euclidiana entre dois pontos de um␣
 ↪sistema cartesiano.
    """
    x1, y1 = ponto1
    x2, y2 = ponto2

    return ((x1-x2)**2 + (y1-y2)**2) ** (1/2)
```

[3]:
```
def gera_coordenadas(n):
    """
    Função que imprime um gráfico contendo n pontos,
    retornando uma lista com as coordenadas dos n pontos no sistema cartesiano
    """
    coords, xa, ya = [], [], []
    for i in range(n):
        x, y = random.randint(-10, 10), random.randint(-10, 10)
        coords += [(x, y)]
        xa += [x]
        ya += [y]
    plt.plot(xa, ya, 'ro')
    plt.axis([-10,10,-10,10])
    plt.show


    return coords
```

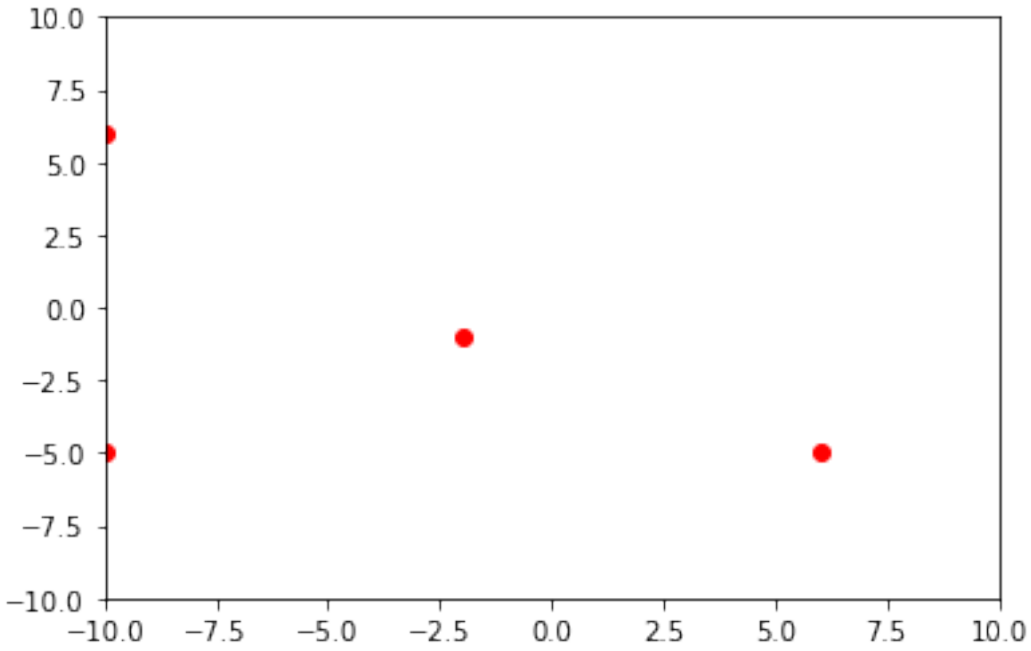
[4]:
```
def pcv(c):
    """
    Recebe uma lista contendo tuplas com coordenadas e
    Imprime as coordenadas das cidades e retorna a combinação
    de cidades que possui a menor distância percorrida e o valor
    da soma
```

```
    """
    k = 1
    for i in range(len(c)):
        print("Cidade {}: ({}, {})".format(k, c[i][0], c[i][1]))
        k += 1
    comb = []
    permutacoes = list(permutations(c, len(c)))
    distancias = []
    soma = 0
    for permutacao in permutacoes:
        k = 0
        while k < len(permutacao)-1:
            soma += distancia(permutacao[k], permutacao[k+1])
            k += 1
        distancias += [soma]
        soma = 0

    # Verificando menor soma de distâncias na variável distancias
    indice_menor_soma = 0
    for s in range(len(distancias)):
        if distancias[s] < distancias[indice_menor_soma]:
            indice_menor_soma = s

    return permutacoes[indice_menor_soma], distancias[indice_menor_soma]
```

[5]:
```
def main(n_cidades):
    cidades_coords = gera_coordenadas(n_cidades)
    comb, menor_soma = %time pcv(cidades_coords)
    print("Melhor caminho: {}".format(comb))
    print("Valor da menor soma: {}".format(menor_soma))
```

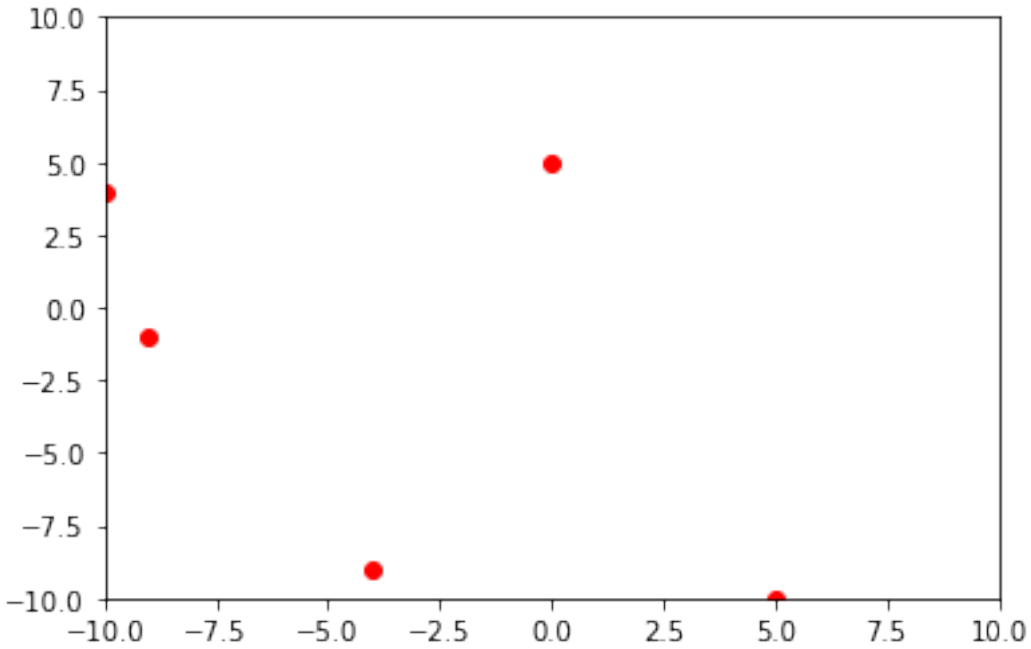
[6]:
```
main(4)
```

```
Cidade 1: (-2, -1)
Cidade 2: (6, -5)
Cidade 3: (-10, -5)
Cidade 4: (-10, 6)
Wall time: 1 ms
Melhor caminho: ((6, -5), (-2, -1), (-10, -5), (-10, 6))
Valor da menor soma: 28.88854381999832
```



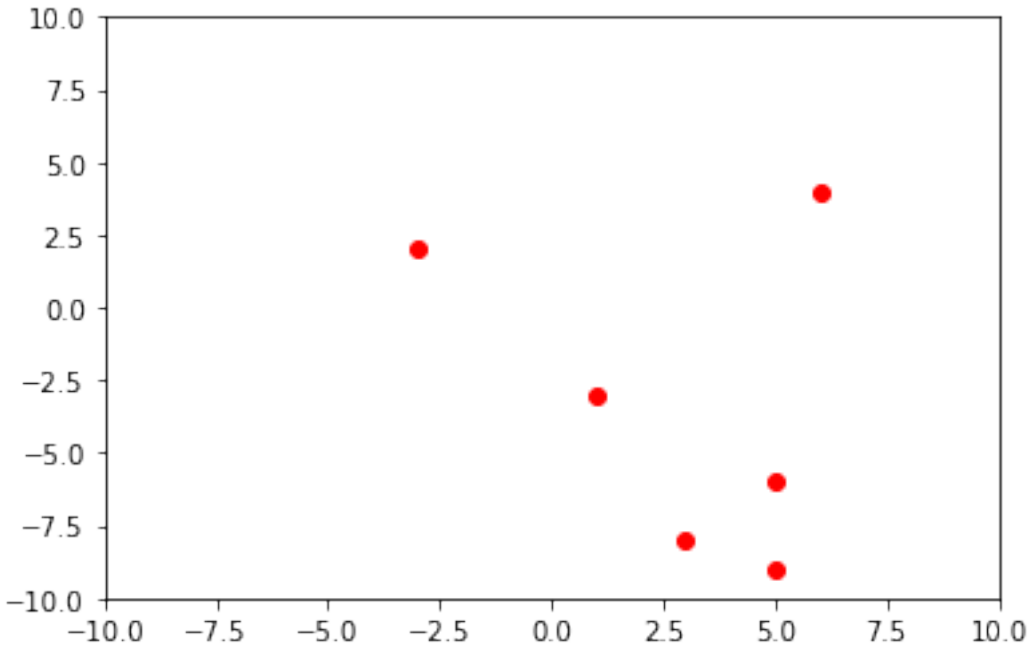
[7]:
```
main(5)
```

```
Cidade 1: (5, -10)
Cidade 2: (-9, -1)
Cidade 3: (-4, -9)
Cidade 4: (0, 5)
Cidade 5: (-10, 4)
Wall time: 996 µs
Melhor caminho: ((5, -10), (-4, -9), (-9, -1), (-10, 4), (0, 5))
Valor da menor soma: 33.638261404907695
```



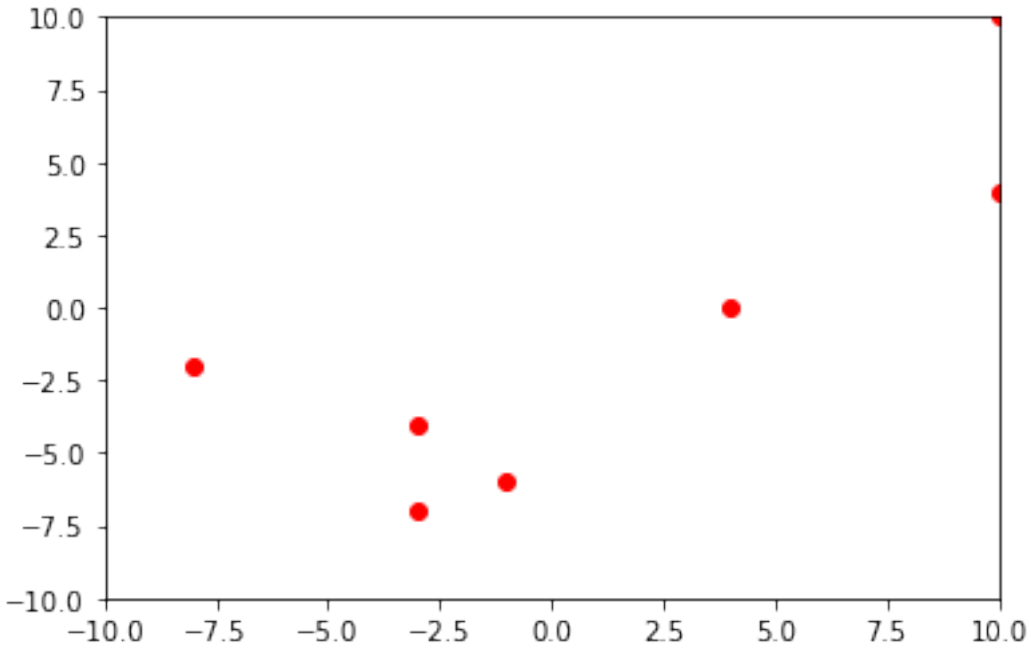
[8]:
```
main(6)
```

```
Cidade 1: (5, -6)
Cidade 2: (5, -9)
Cidade 3: (3, -8)
Cidade 4: (1, -3)
Cidade 5: (6, 4)
Cidade 6: (-3, 2)
Wall time: 6.98 ms
Melhor caminho: ((5, -9), (3, -8), (5, -6), (1, -3), (-3, 2), (6, 4))
Valor da menor soma: 25.687163796971713
```



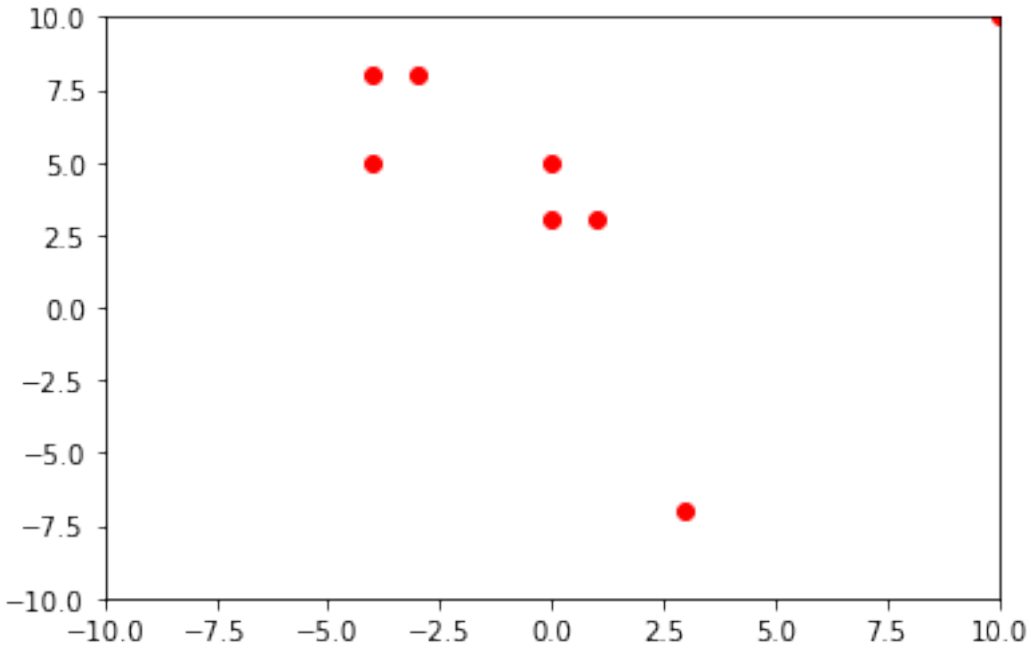
[9]:
```
main(7)
```

```
Cidade 1: (10, 10)
Cidade 2: (4, 0)
Cidade 3: (-1, -6)
Cidade 4: (-3, -4)
Cidade 5: (-8, -2)
Cidade 6: (10, 4)
Cidade 7: (-3, -7)
Wall time: 37.9 ms
Melhor caminho: ((10, 10), (10, 4), (4, 0), (-1, -6), (-3, -7), (-3, -4), (-8,
-2))
Valor da menor soma: 31.642585011468928
```



[10]:
```
main(8)
```

```
Cidade 1: (0, 3)
Cidade 2: (3, -7)
Cidade 3: (10, 10)
Cidade 4: (1, 3)
Cidade 5: (-3, 8)
Cidade 6: (-4, 8)
Cidade 7: (0, 5)
Cidade 8: (-4, 5)
Wall time: 299 ms
Melhor caminho: ((3, -7), (1, 3), (0, 3), (0, 5), (-4, 5), (-4, 8), (-3, 8),
(10, 10))
Valor da menor soma: 34.350985465151474
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0