teoremaKreinMilman

October 11, 2023

1 O teorema de Krein-Milmann

Prof. Doherty Andrade – apenas para uso didático

[1]:
```
myst_enable_extensions = [
    "amsmath",
    "amssymb",
    "attrs_inline",
    "colon_fence",
    "deflist",
    "dollarmath",
    "fieldlist",
    "html_admonition",
    "html_image",
    "linkify",
    "replacements",
    "smartquotes",
    "strikethrough",
    "substitution",
    "tasklist",
]
```

Seja X um espaço vetorial sobre $\mathbb{R}$ e $S \subset X$. Dizemos que S é convexo se, dados quaisquer dois pontos x_1 e x_2 de S, o segmento de reta que os une está inteiramente contido em S. Em outras palavras:

para quaisquer x_1 e x_2 em S e $\lambda \in [0,1]$ tem-se $\lambda x_1 + (1-\lambda)x_2 \in S$.

Tais combinações lineares são chamadas de combinações convexas de dois elementos.

Vamos nos concentrar nos conjuntos convexos do espaço $\mathbb{R}^n$.

Proposição:

(a) Se S_1 e S_2 são conjuntos convexos, então $S_1 \cap S_2$ é convexo.

(b) Se $S \subset V_1$ é convexo e $T : V_1 \to V_2$ é uma transformação linear entre espaços vetoriais, então $T(S) \subset V_2$ é convexo.

(c) Se $Y \subset V_2$ é convexo, então $S = T^{-1}(Y)$ é convexo, onde T é transformação linear.

A envoltória convexa, ou o casco convexo, de um conjunto X é o menor conjunto convexo S que contém X. Por menor conjunto entende-se que S a interseção de todos conjuntos convexos que contém X.

Outra caracterização para a envoltória convexa é dada por

$$S = \{\lambda x + (1-\lambda)y; x, y \in X, \lambda \in [0,1]\}.$$

Graficamente, a envoltoria convexa S de um conjunto X é o conjunto S de todos os pontos que podem ser escritos como uma combinação convexa de dois elementos quaisquer de X.

Teorema 1: Sejam $x_1, x_2, \ldots, x_m$ pontos do espaço $\mathbb{R}^n$. Qualquer conjunto convexo que contenha esses pontos tem que conter todas as combinações lineares

$$\lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_n x_n$$

onde $\lambda_i \in [0,1]$ e $\lambda_1 + \lambda_2 + \ldots + \lambda_n = 1$.

Tais combinações lineares são chamadas de combinações convexas.

A demonstração mais conhecida utiliza indução e deixamos como exercício. Neste caso, em algum momento da sua demonstração você vai precisar supor que se $\lambda_n \neq 1$, então a combinação linear acima é igual a

$$(1-\lambda_n)\left(\frac{\lambda_1}{1-\lambda_n}x_1 + \cdots + \frac{\lambda_{n-1}}{1-\lambda_n}x_{n-1}\right) + \lambda_n x_n.$$

Teorema 2: Sejam $x_1, x_2, \ldots, x_m$ pontos do espaço $\mathbb{R}^n$. O conjunto de todas as combinações lineares

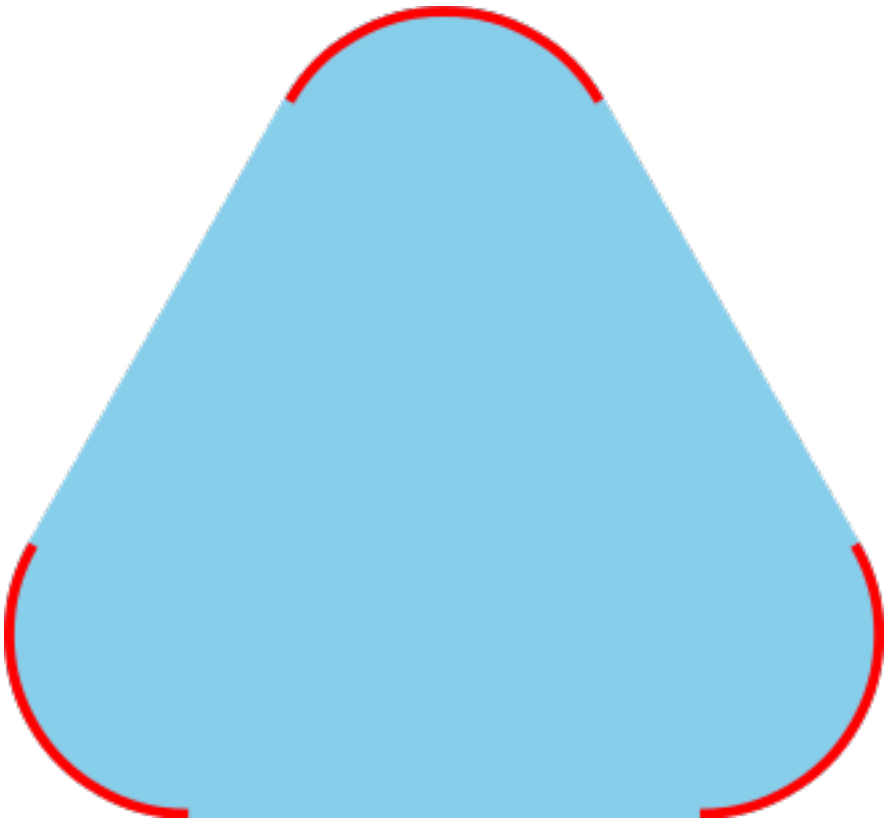
$$\lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_n x_n$$

onde $\lambda_i \in [0,1]$ e $\lambda_1 + \lambda_2 + \ldots + \lambda_n = 1$ é um conjunto convexo.

A demonstração é simples e fica como exercício.

Como consequência desses dois teoremas, concluímos que o conjunto das combinações lineares descritas (combinação convexa) nos teoremas é o menor conjunto convexo contendo os elementos $x_1, x_2, \ldots, x_m$.

Dado um conjunto X o conjunto de todas as combinações convexas de elementos de X é chamado de envolvotória convexa, casco convexo ou fecho convexo.



O fecho convexo, o casco convexo ou envoltoria convexa do conjunto vermelho é o conjunto azul.

2 Pontos extremos

Seja S um conjunto convexo e $x_0 \in S$. Dizemos que x_0 é um ponto extremo de S se não existirem x_1 e x_2 elementos de S, como $x_1 \neq x_2$, tais que x_0 possa ser escrito da forma de combinação convexa.

Isto é, não é possível escrever

$$x_0 = \lambda x_1 + (1 - \lambda)x_2$$

com $0 < \lambda < 1$.

Em outras palavras, x_0 não pode pertencer a um segmento de reta inteiramente contido em S, a não ser que, x_0 seja extremo do segmento de reta.

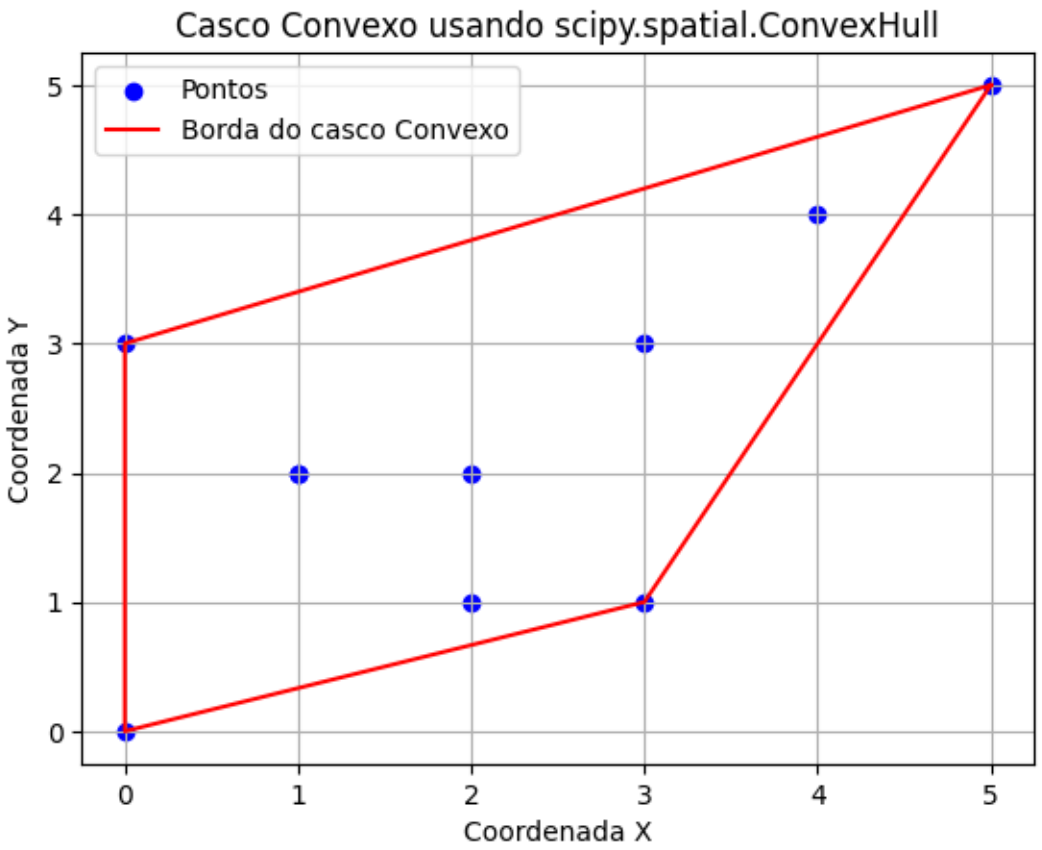
Teorema 3: Todo conjunto convexo e compacto S tem um ponto extremo.

Teorema (Krein-Milmann): Seja S um compacto convexo e K o conjunto de seus pontos extremos. Então S é o menor conjunto convexo fechado que contém todos os elementos de K. Isto é, S é a intersecção de todos os conjuntos convexos e fechados contendo K.

Em palavras, todo convexo compato é gerado por combinações convexas de seus pontos extremos.

No código em Python abaixo, fornecemos alguns pontos do plano e o código desenha a envoltoria convexa do conjunto de pontos.

[2]:
```python
# Author: Doherty Andrade
# Mail: doherty200@hotmail.com
# Script: Compute the Convex Hull of a set of points

import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull

# Exemplo de uso:
if __name__ == "__main__":
    points = [(0, 3), (2, 1), (2, 2), (4, 4), (0, 0), (1, 2), (3, 1), (3, 3),␣
 ↪(5,5), (1,2)]

    # Calcula a casca convexa usando scipy.spatial.ConvexHull
    hull = ConvexHull(points)

    # Extrai as coordenadas x e y dos pontos para o plot.
    x = [point[0] for point in points]
    y = [point[1] for point in points]

    # Extrai as coordenadas x e y da casca convexa para o plot.
    convex_x = [points[i][0] for i in hull.vertices]
    convex_y = [points[i][1] for i in hull.vertices]

    # Plota os pontos originais e o casco convexo.
    plt.figure()
    plt.scatter(x, y, color='blue', label='Pontos')
    plt.plot(convex_x + [convex_x[0]], convex_y + [convex_y[0]], color='red',␣
 ↪label='Borda do casco Convexo')
    plt.xlabel('Coordenada X')
    plt.ylabel('Coordenada Y')
    plt.title('Casco Convexo usando scipy.spatial.ConvexHull')
    plt.legend()
    plt.grid(True)
    plt.show()
```



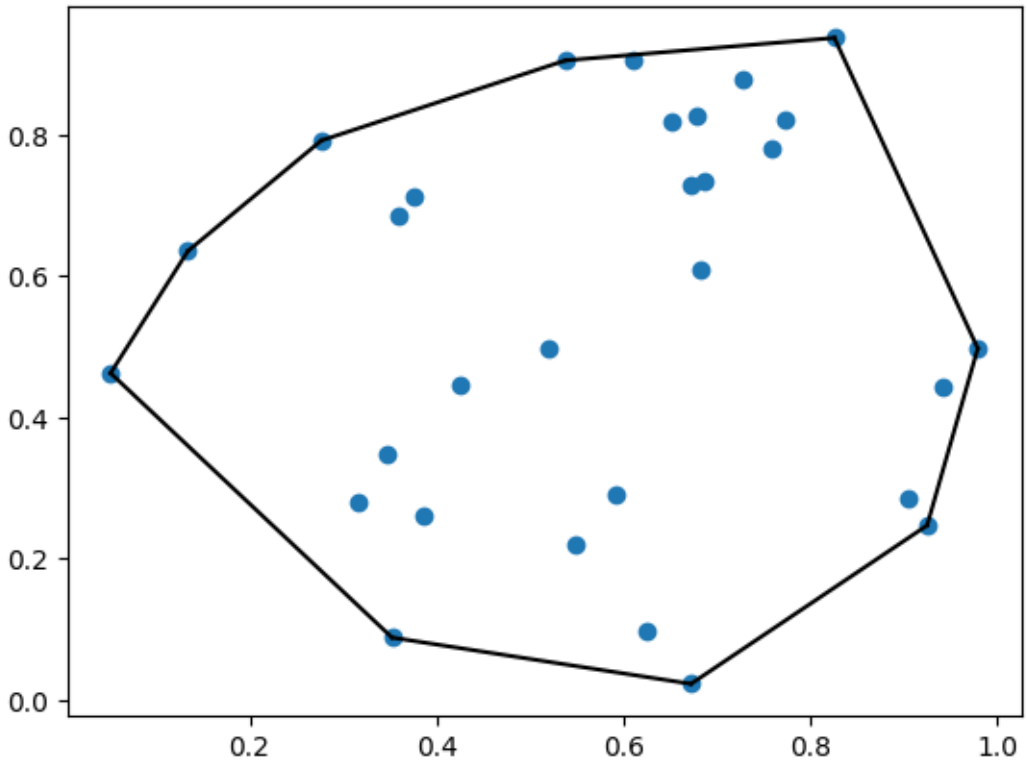
2.1 Outros códigos

O código abaixo gera aletoriamente 30 pontos no plano com coordenadas no intervalo $[0, 1]$ e plota a envoltória do conjunto.

Note que a região interna também faz parte da envoltória convexa do conjunto.

[2]:
```
from scipy.spatial import ConvexHull, convex_hull_plot_2d
import numpy as np
rng = np.random.default_rng()
points = rng.random((30, 2))   # 30 random points in 2-D
hull = ConvexHull(points)
```

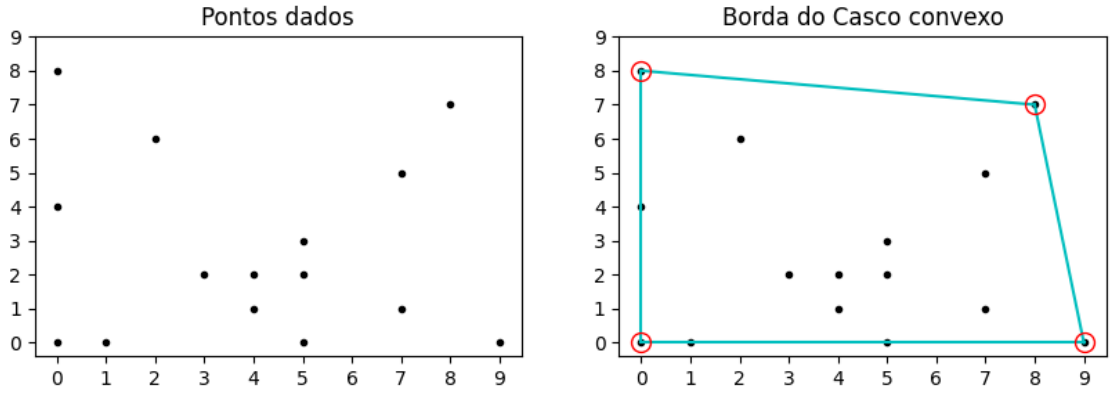
[3]:
```
import matplotlib.pyplot as plt
plt.plot(points[:,0], points[:,1], 'o')
for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'k-')
```



Do mesmo modo o codigo abaixo gera pontos do plano aleatoriamente e em seguida plota a envoltoria convexa.

[4]:
```
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
import numpy as np

points = np.random.randint(0, 10, size=(15, 2))  # Random points in 2-D

hull = ConvexHull(points)

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 3))

for ax in (ax1, ax2):
    ax.plot(points[:, 0], points[:, 1], '.', color='k')
    if ax == ax1:
        ax.set_title('Pontos dados')
    else:
        ax.set_title('Borda do Casco convexo')
        for simplex in hull.simplices:
            ax.plot(points[simplex, 0], points[simplex, 1], 'c')
```

```
        ax.plot(points[hull.vertices, 0], points[hull.vertices, 1], 'o',␣
↪mec='r', color='none', lw=1, markersize=10)
    ax.set_xticks(range(10))
    ax.set_yticks(range(10))
plt.show()
```



2.2 O algoritmo de Graham

O algoritmo de Graham é usado para encontrar o casco convexo de um conjunto de pontos em um plano 2D.

O casco convexo de um conjunto de pontos é o menor conjunto convexo que engloba todos os pontos do conjunto. Em outras palavras, é uma figura convexa que cobre todos os pontos do conjunto.

O algoritmo de Graham opera da seguinte forma:

1. Encontre o ponto com a menor coordenada y (ponto mais abaixo).
2. Ordenar todos os outros pontos com base em seus ângulos polares em relação ao ponto encontrado no passo 1. O ângulo polar é medido a partir do eixo x e aumenta no sentido anti-horário.
3. Percorra a lista ordenada de pontos para construir a casca convexa.
    - Inicialmente, adicione os dois primeiros pontos da lista à casca convexa.
    - Para cada ponto subsequente, verifique se ele faz uma curva à esquerda com os dois últimos pontos da casca convexa (se fizer uma curva à direita, esse ponto está dentro da casca convexa e não é considerado). Se fizer uma curva à esquerda, adicione-o à casca convexa.
    - Repita o passo anterior até que todos os pontos tenham sido considerados.
    - No final, você terá a casca convexa do conjunto de pontos.

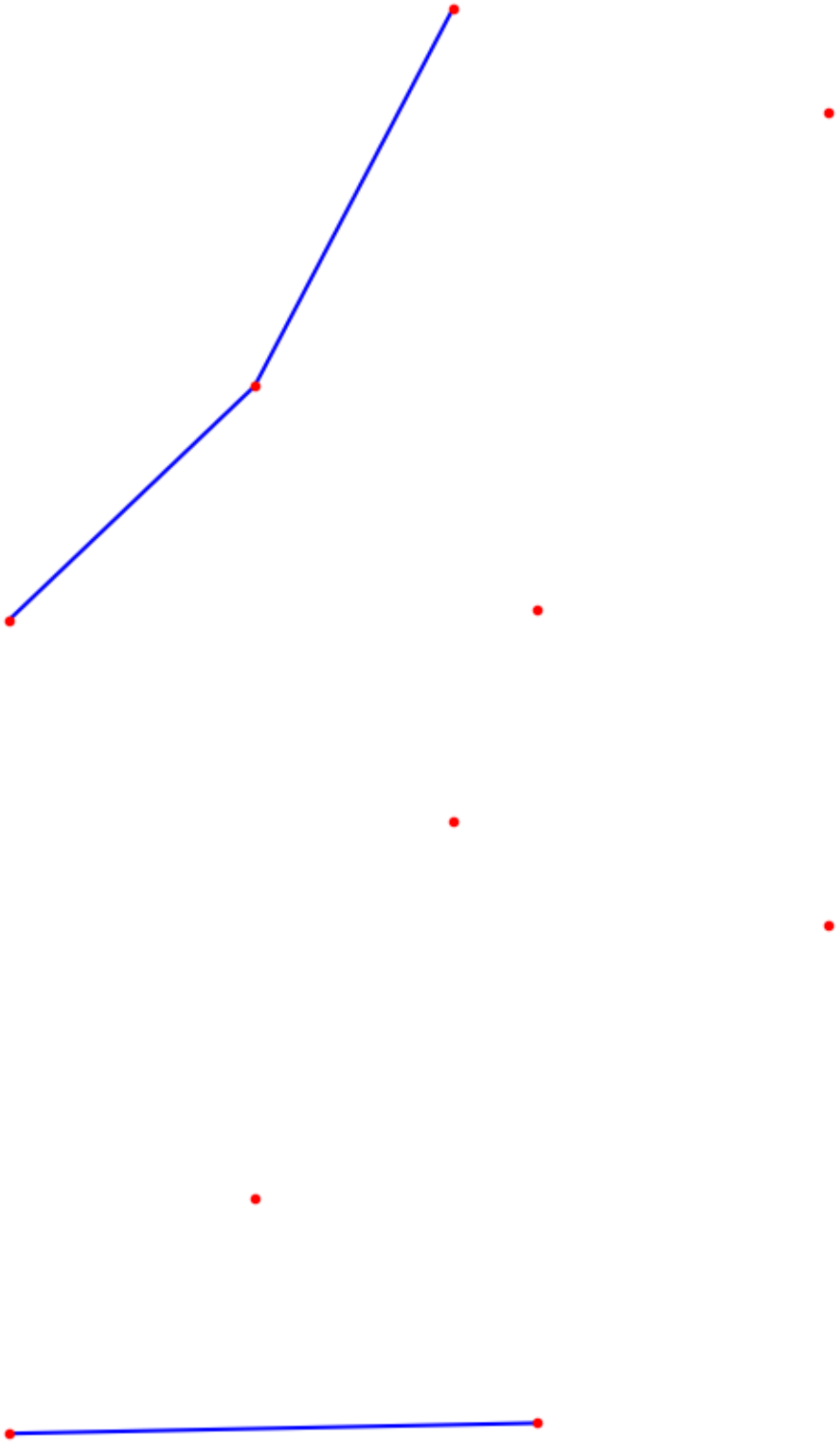
O algoritmo de Graham é eficiente e pode ser implementado com uma complexidade de tempo de $O(n \log n)$, onde n é o número de pontos no conjunto. No entanto, pode enfrentar problemas de desempenho em casos de pontos colineares, já que a ordenação dos ângulos polares pode não ser

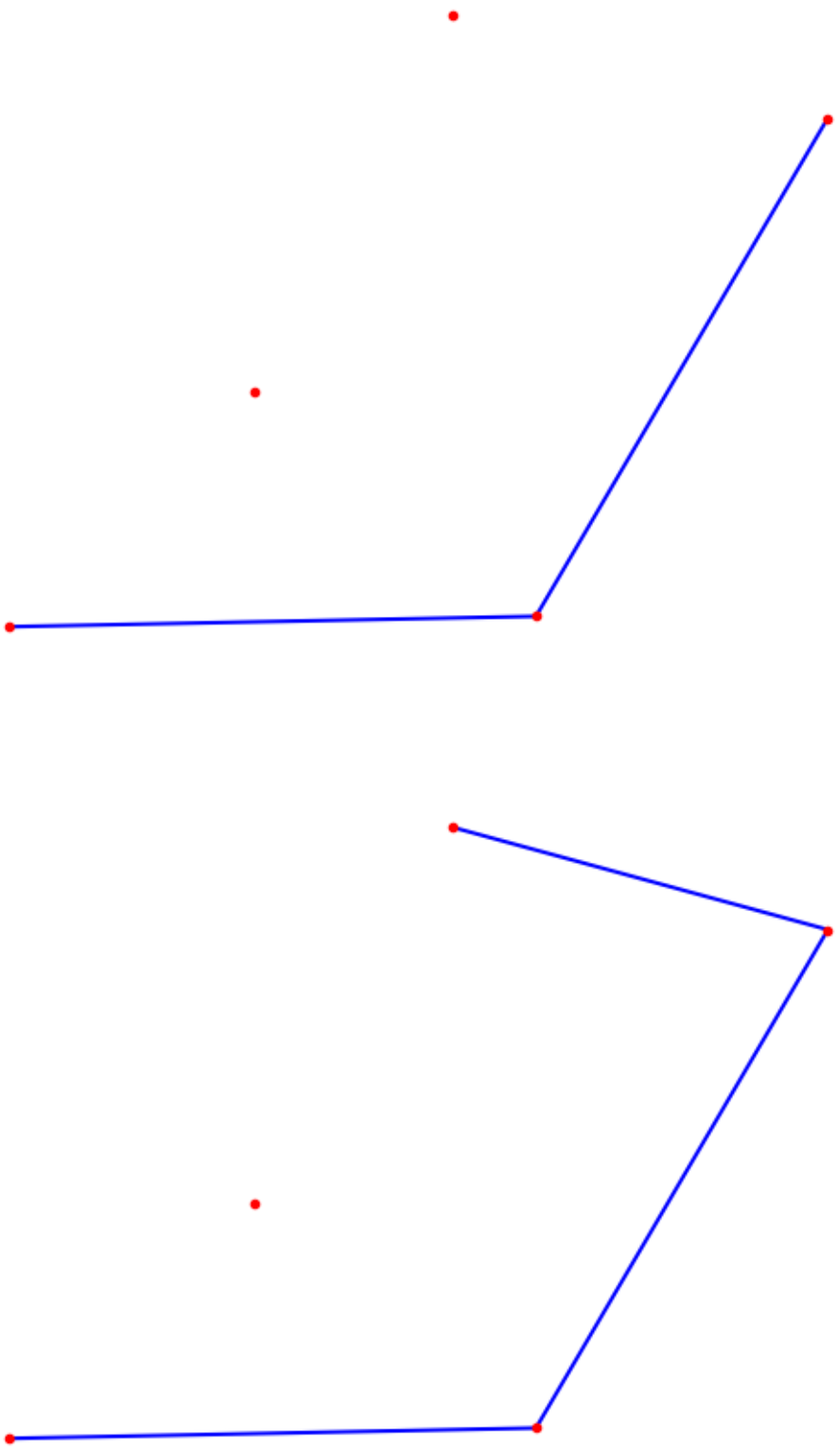
única e pode levar a um pior caso de $O(n^2)$ na complexidade do algoritmo. Portanto, é importante garantir que os pontos não estejam colineares para obter o melhor desempenho do algoritmo.

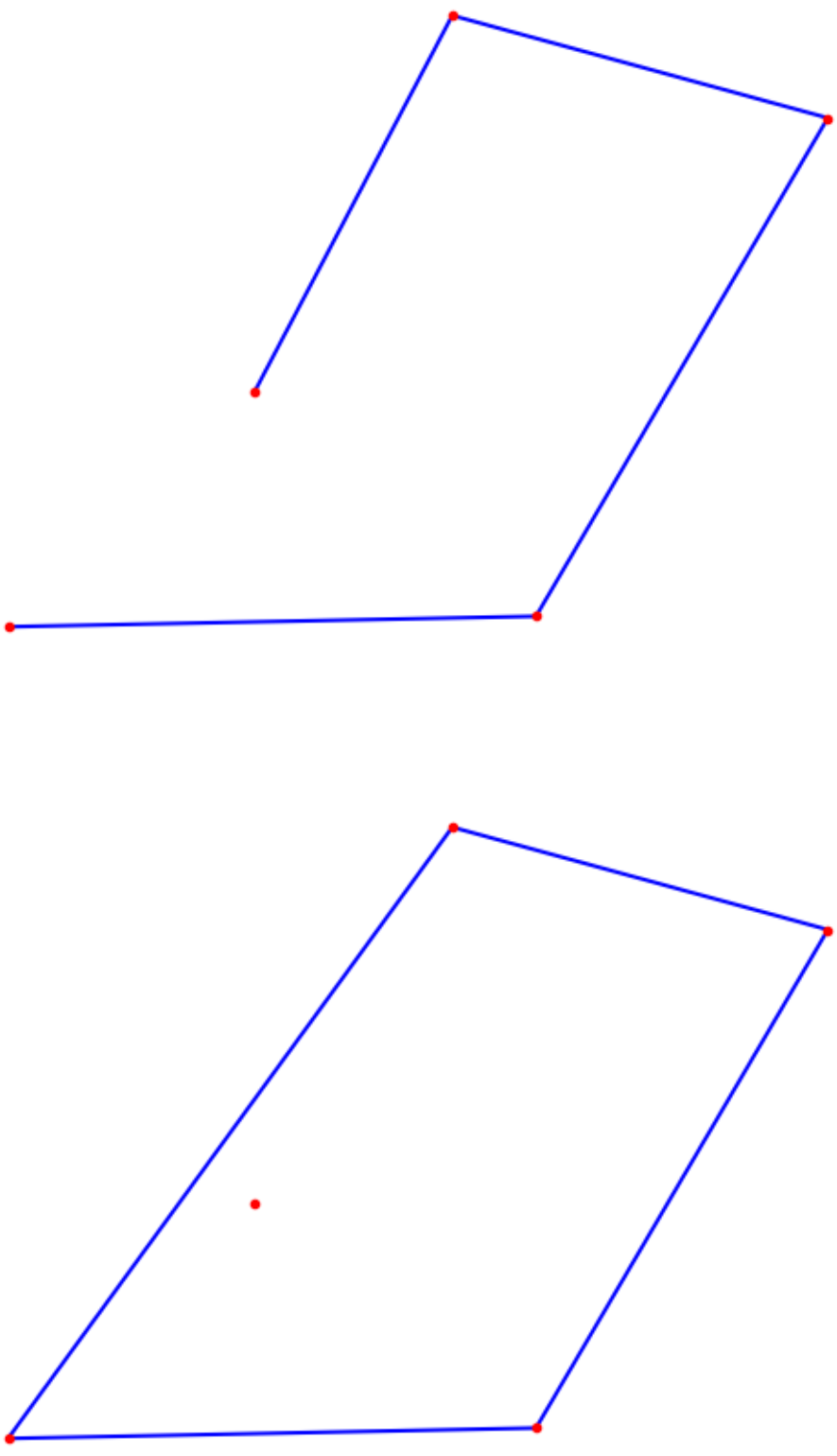
[6]:
```
# Author: Rodolfo Ferro
# Mail: ferro@cimat.mx
# Script: Compute the Convex Hull of a set of points using the Graham Scan␣
 ↪(Plotting each step!)

import sys
import numpy as np
import matplotlib.pyplot as plt

# Function to know if we have a CCW turn
def RightTurn(p1, p2, p3):
        if (p3[1]-p1[1])*(p2[0]-p1[0]) >= (p2[1]-p1[1])*(p3[0]-p1[0]):
                return False
        return True

# Main algorithm:
def GrahamScan(P):
        P.sort()                        # Sort the set of points
        P = np.array(P)                         # Convert the list to numpy array
        plt.figure()                        # Create a new fig
        L_upper = [P[0], P[1]]                  # Initialize the upper part
        # Compute the upper part of the hull
        for i in range(2,len(P)):
                L_upper.append(P[i])
                while len(L_upper) > 2 and not␣
 ↪RightTurn(L_upper[-1],L_upper[-2],L_upper[-3]):
                        del L_upper[-2]
                L = np.array(L_upper)
                plt.clf()                # Clear plt.fig
                plt.plot(L[:,0],L[:,1], 'b-', picker=5)         # Plot lines
                plt.plot(P[:,0],P[:,1],".r")                # Plot points
                plt.axis('off')                 # No axis
                plt.show(block=False)         # Close plot
                plt.pause(0.0000001)         # Mini-pause before closing plot
        L_lower = [P[-1], P[-2]]         # Initialize the lower part
        # Compute the lower part of the hull
        for i in range(len(P)-3,-1,-1):
                L_lower.append(P[i])
                while len(L_lower) > 2 and not␣
 ↪RightTurn(L_lower[-1],L_lower[-2],L_lower[-3]):
                        del L_lower[-2]
                L = np.array(L_upper + L_lower)
                plt.clf()                # Clear plt.fig
                plt.plot(L[:,0],L[:,1], 'b-', picker=5)    # Plot lines
```

```
                plt.plot(P[:,0],P[:,1],".r")              # Plot points
                plt.axis('off')          # No axis
                plt.show(block=False)    # Close plot
                plt.pause(0.0000001)         # Mini-pause befor closing plot
        del L_lower[0]
        del L_lower[-1]
        L = L_upper + L_lower                  # Build the full hull
        plt.axis('off')
        plt.show()
        return np.array(L)

def main():
        try:
                N = int(sys.argv[1])
        except:
                N = int(input("Entre com N: "))

        # By default we build a random set of N points with coordinates in␣
 ↪[-300,300)x[-300,300):
        P = [(np.random.randint(-200,200),np.random.randint(-200,200)) for i in␣
 ↪range(N)]
        L = GrahamScan(P)

if __name__ == '__main__':
  main()
```

```
Entre com N: 5
```



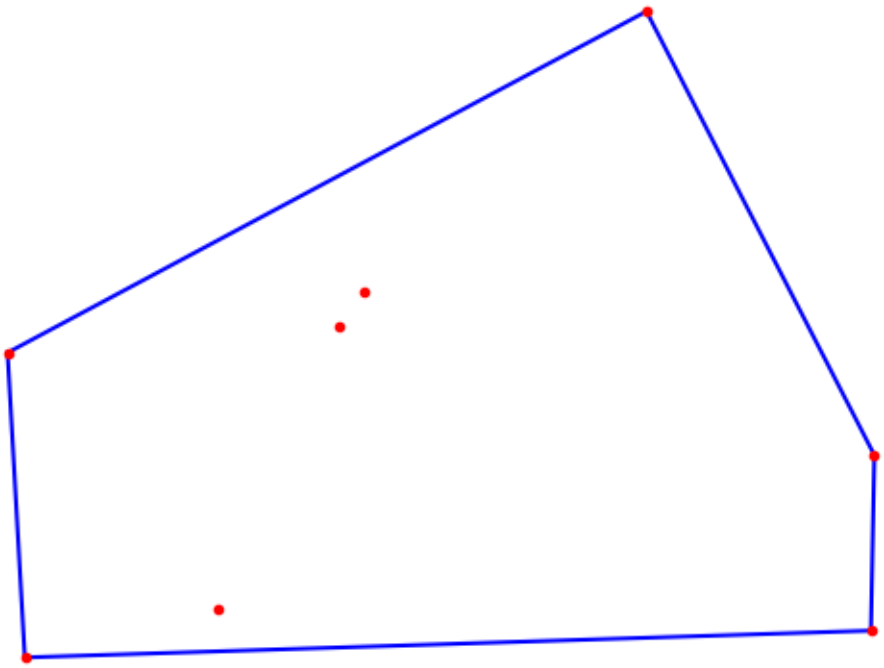




[5]:
```
# Author: Rodolfo Ferro
# Mail: ferro@cimat.mx
# Script: Compute the Convex Hull of a set of points using the Graham Scan

import sys
import numpy as np
import matplotlib.pyplot as plt

# Function to know if we have a CCW turn
def RightTurn(p1, p2, p3):
        if (p3[1]-p1[1])*(p2[0]-p1[0]) >= (p2[1]-p1[1])*(p3[0]-p1[0]):
                return False
        return True

# Main algorithm:
def GrahamScan(P):
        P.sort()                        # Sort the set of points
        L_upper = [P[0], P[1]]                # Initialize upper part
        # Compute the upper part of the hull
        for i in range(2,len(P)):
                L_upper.append(P[i])
                while len(L_upper) > 2 and not␣
 ↪RightTurn(L_upper[-1],L_upper[-2],L_upper[-3]):
```

```
                        del L_upper[-2]
        L_lower = [P[-1], P[-2]]        # Initialize the lower part
        # Compute the lower part of the hull
        for i in range(len(P)-3,-1,-1):
                L_lower.append(P[i])
                while len(L_lower) > 2 and not␣
↪RightTurn(L_lower[-1],L_lower[-2],L_lower[-3]):
                        del L_lower[-2]
        del L_lower[0]
        del L_lower[-1]
        L = L_upper + L_lower                # Build the full hull
        return np.array(L)

def main():
        try:
                N = int(sys.argv[1])
        except:
                N = int(input("Entre com N: "))

        # By default we build a random set of N points with coordinates in␣
↪[0,300)x[0,300):
        P = [(np.random.randint(0,300),np.random.randint(0,300)) for i in␣
↪range(N)]
        L = GrahamScan(P)
        P = np.array(P)

        # Plot the computed Convex Hull:
        plt.figure()
        plt.plot(L[:,0],L[:,1], 'b-', picker=5)
        plt.plot([L[-1,0],L[0,0]],[L[-1,1],L[0,1]], 'b-', picker=5)
        plt.plot(P[:,0],P[:,1],".r")
        plt.axis('off')
        plt.show()

if __name__ == '__main__':
        main()
```

```
Entre com N: 8
```



Referências

[1] Serge Lang. Algebra Linear.

[ ]: