Disciplina INF2604 – Geometria Computacional

RELATÓRIO DE TRABALHO:
Fecho Convexo

Aluno: Gustavo Coelho

1. Objetivo

O objetivo deste relatório é detalhar os conceitos teóricos e a abordagem prática usada para resolver o problema de definição do fecho mínimo de uma nuvem de pontos através do algoritmo “Gift Wrapping”. Assim, veremos como são feitos os cálculos para a definição dos pontos dos fechos, sua implementação em linguagem Python e uma análise da complexidade do tempo de execução de acordo com o tamanho da entrada.

2. Método

O método escolhido (“Gift Wrapping”) é de relativa simples implementação. O primeiro passo do algoritmo é encontrar o ponto mais inferior da nuvem, ou seja, o ponto com a menor coordenada y. Isso é facilmente feito percorrendo a nuvem por completo, com um custo de execução proporcional a $O_{(n)}$. Essa propriedade (menor coordenada y) nos garante que esse ponto necessariamente faz parte do fecho convexo. Portanto, começaremos a traçar o fecho a partir dele. O algoritmo então irá definir os próximos pontos encontrando o menor ângulo θ entre a reta formada pelo ponto atual e o ponto anterior do fecho e a reta formada entre o ponto atual e o próximo ponto (no sentido anti-horário), conforme ilustrado abaixo:

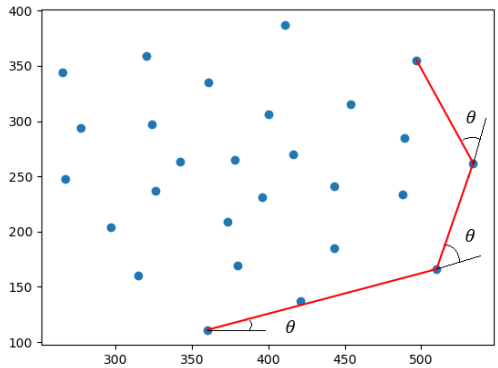

Figura 1: Identificação dos menores ângulos θ.

Para o primeiro ponto encontrado, consideramos a reta de comparação como sendo uma reta horizontal partindo deste ponto.

Para definir os ângulos θ, consideramos os pontos a, b e c, onde b é o ponto atual definido como parte do fecho, c é o ponto do fecho anterior à b e a sendo qualquer outro ponto que não seja igual a b ou c:

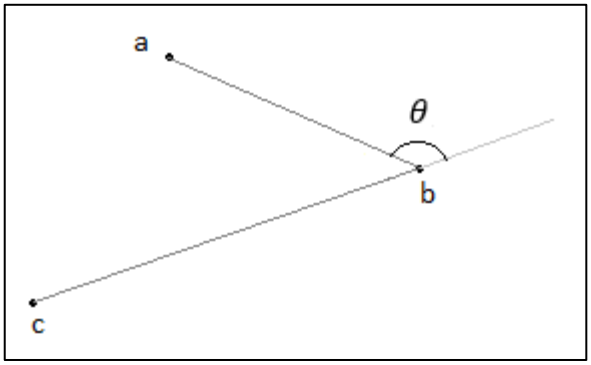

Figura 2: Identificação de θ a partir dos pontos a, b e c.

Assumimos inicialmente que o ponto a possui coordenada "à esquerda" de $\overline{cb}$. Neste caso, podemos definir θ como o ângulo formado pelos vetores $\overrightarrow{(b-c)}$ e $\overrightarrow{(a-b)}$. Portanto, podemos calcular θ da seguinte forma:

$$\theta = \cos^{-1}(\vec{u}.\vec{v}), \text{ onde } \vec{u} = \left(\frac{\overrightarrow{(b-c)}}{\|\overrightarrow{(b-c)}\|}\right), \quad \vec{v} = \left(\frac{\overrightarrow{a-b}}{\|\overrightarrow{a-b}\|}\right)$$

Ou seja:

$$\theta = \cos^{-1}\left(u_x v_x + u_y v_y\right)$$

Onde:

$$\vec{u} = \left(\frac{b_x - c_x}{\sqrt{(b_x - c_x)^2 + \left(b_y - c_y\right)^2}}, \frac{b_y - c_y}{\sqrt{(b_x - c_x)^2 + \left(b_y - c_y\right)^2}}\right)$$

$$\vec{v} = \left(\frac{a_x - b_x}{\sqrt{(a_x - b_x)^2 + \left(a_y - b_y\right)^2}}, \frac{a_y - b_y}{\sqrt{(a_x - b_x)^2 + \left(a_y - b_y\right)^2}} \right)$$

Porém, para casos em que o ponto a situa-se “à direita” de $\overline{cb}$, este cálculo não resultará no ângulo desejado, conforme vemos na figura abaixo:

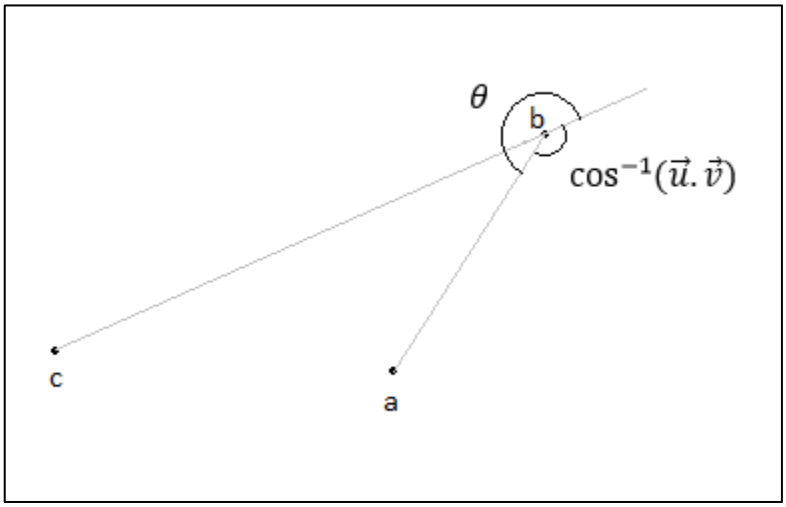

Figura 3: Ponto a “à direita” de $\overline{cb}$.

Para estes casos, o angulo θ é definido como:

$$\theta = 2\pi - \cos^{-1}(\vec{u}.\vec{v})$$

Portanto, antes de definir o ângulo, devemos definir se o ponto a está “à direita” ou “à esquerda” de $\overline{cb}$. Em outras palavras, devemos definir a orientação do triângulo formado por b, c e a:

$$< b, c, a > = \begin{vmatrix} b_x & b_y & 1 \\ c_x & c_y & 1 \\ a_x & a_y & 1 \end{vmatrix} = b_x c_y + b_y a_x + c_x a_y - a_x c_y - a_y b_x - c_x b_y$$

Caso a orientação seja positiva, consideramos o ponto a “à esquerda”, caso negativa, consideramos o ponto a “à direita”.

Para cada ponto do fecho, este cálculo é feito para todos os outros pontos até que o menor ângulo seja encontrado. Portanto, o custo de execução esperado será proporcional a $O_{(nh)}$, onde h é o número de pontos no fecho.

3. Implementação

Implementando o algoritmo em linguagem Python, e usando como entrada conjuntos de pontos pré-definidos, obtemos os seguintes resultados:

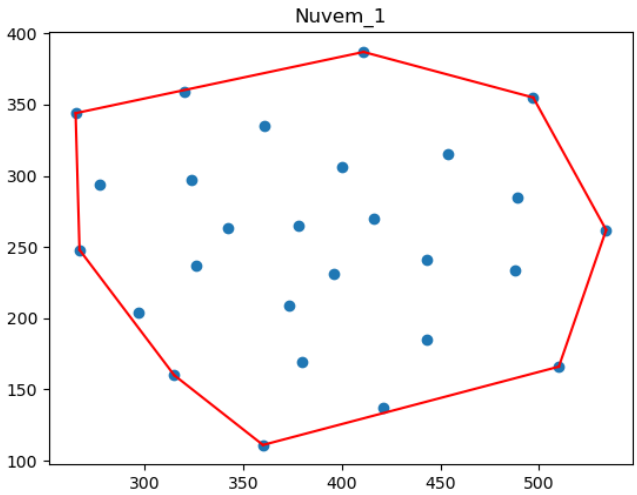

Figura 4: Fecho convexo para Nuvem_1.

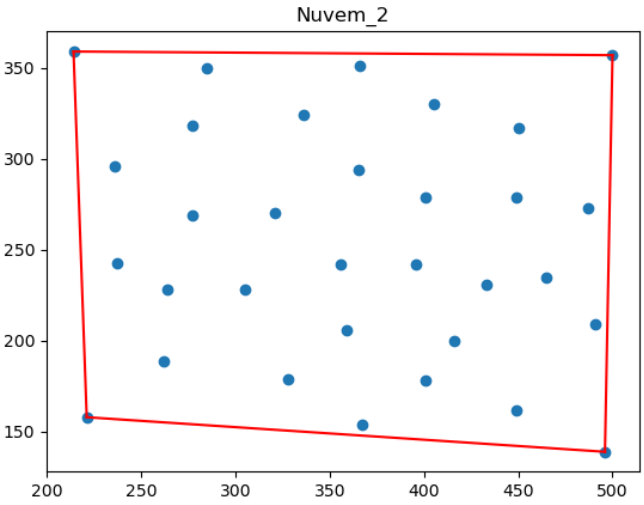

Figura 5: Fecho convexo para Nuvem_2.

Também é possível implementar o algoritmo para um conjunto de pontos criados de forma aleatória, com distribuição normal. Abaixo vemos o resultado para uma nuvem de 1000 pontos:

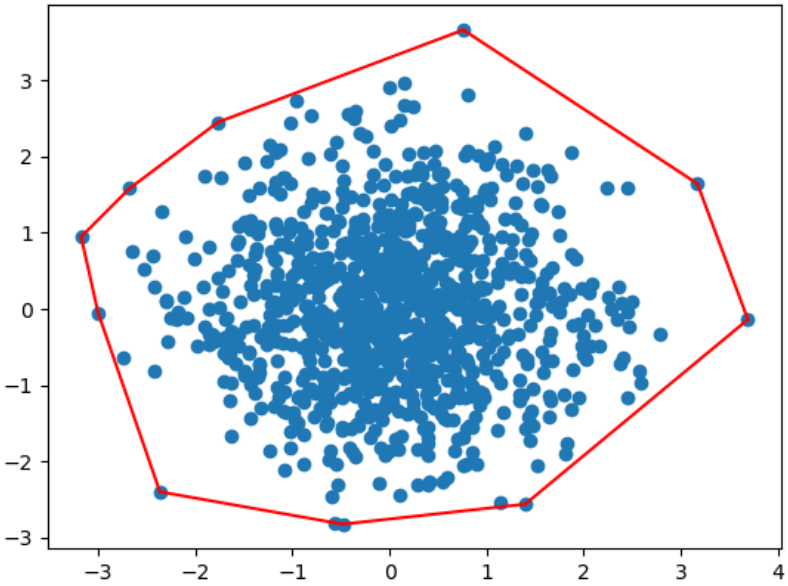

Figura 6: Fecho convexo para 1000 pontos aleatórios.

Abaixo podemos analisar o tempo de execução para diferentes tamanhos de entrada:

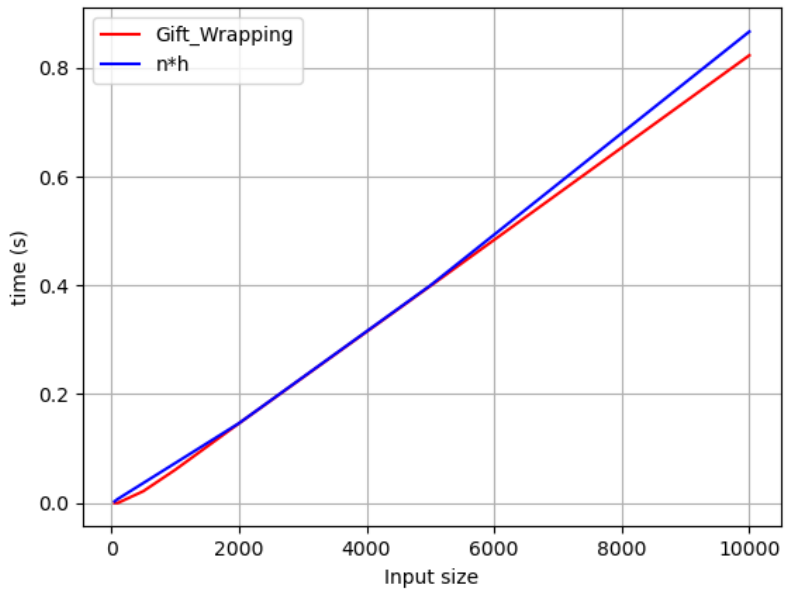

Figura 7: Tempo de execução de acordo com tamanho de entrada.

4. Conclusão

O algoritmo “Gift Wrapping” para identificação do fecho convexo fornece uma solução relativamente simples comparado à outros algoritmos de mesma função. Conforme esperado, o tempo de execução do algoritmo proposto de acordo com a entrada é proporcional a $O_{(nh)}$. Isso pode resultar em um algoritmo muito eficiente, principalmente para conjuntos com muitos pontos e poucos pontos situados no feche. É também uma boa alternativa ao algoritmo “força bruta”, que resultaria em uma complexidade $O_{(n^2)}$. Porém, para casos especiais onde existem muitos pontos no feche, o algoritmo “Gift Wrapping” pode não ser a melhor alternativa.