Exercício 3 - GC MDCC 2021.1

Carolina Herbster

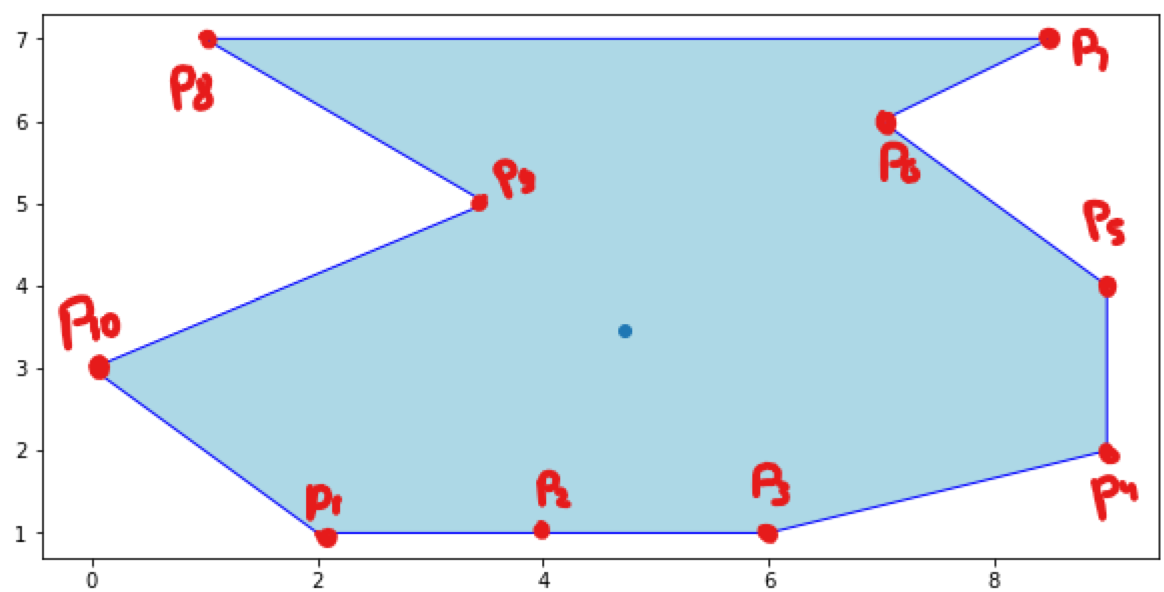
Questão 01

Podemos realizar a redução da ordenação ao fecho convexo da seguinte maneira. Dados os números reais $x_1, x_2, \ldots, x_n$, formamos o conjunto $C = \{p_1, \ldots, p_n\}$, tal que $p_i = (x_i, x_i^2)$, e executamos o algoritmo do fecho convexo. Cada um dos pi é vértice do fecho convexo, e o algoritmo do fecho os ordena circularmente de acordo com as suas abscissas x_i. Podemos então obter o ponto de menor abcissa e ler as abscissas seguintes em ordem.

Questão 02

Item a

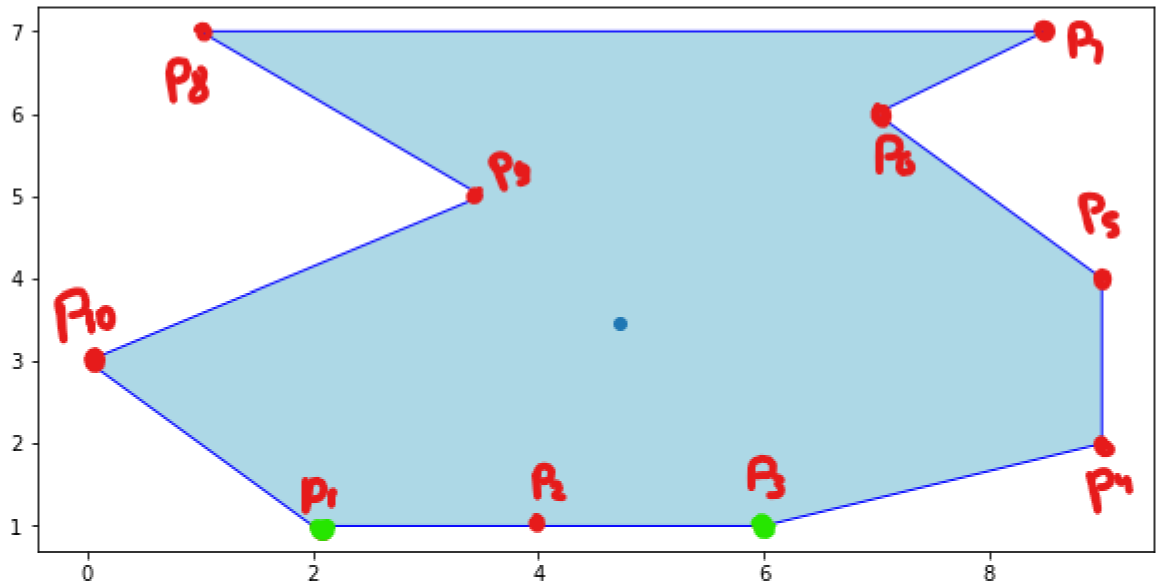
O polígono estrelado dos pontos dados é o seguinte:



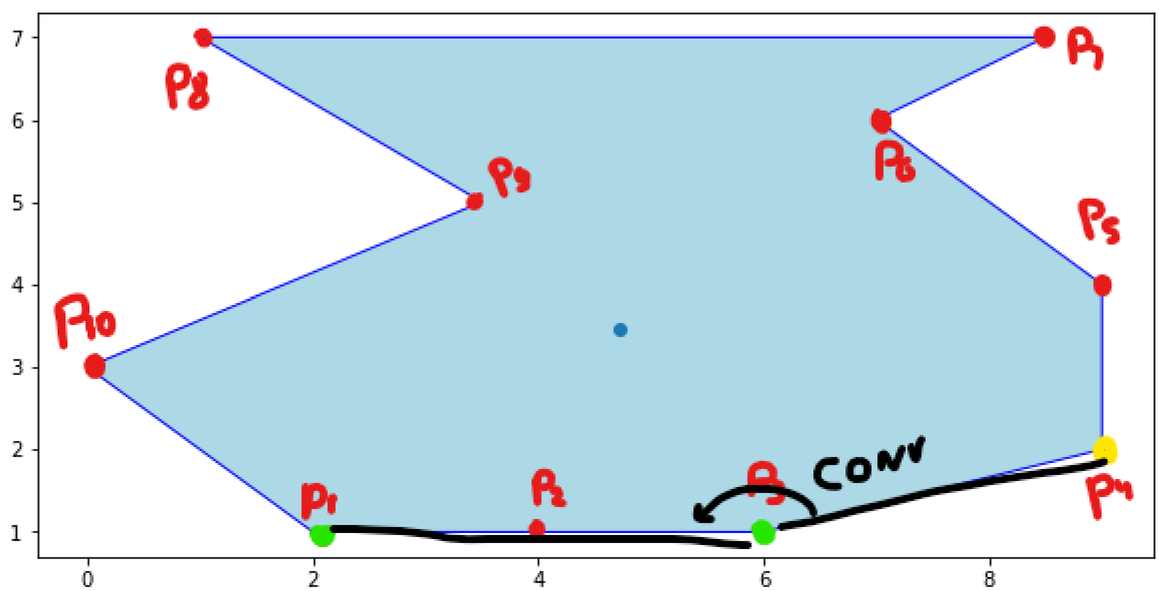
Item b

1. Primeiro, ordenamos os pontos de acordo com o àngulo orientado em relação a p1. Se dois pontos tem o mesmo ângulo, como p2 e p3, descartamos o mais próximo, no caso p2. Obtemos a ordem: p1,p3,p4,p5,p7,p6,p9,p8,p10

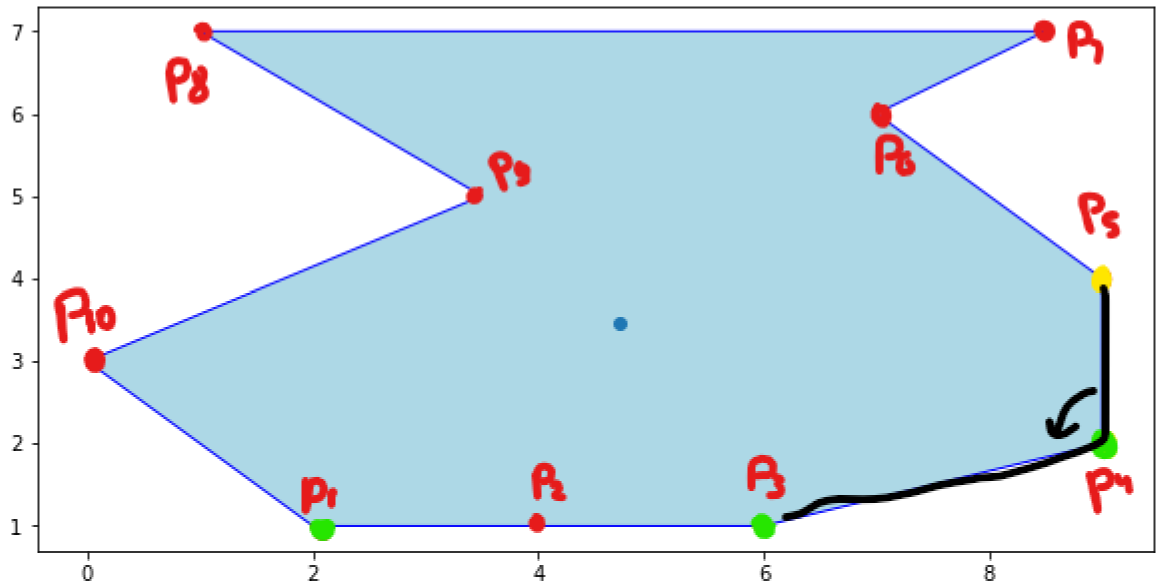
2. Iniciamos o fecho com os dois primeiros pontos da lista, p1 e p3



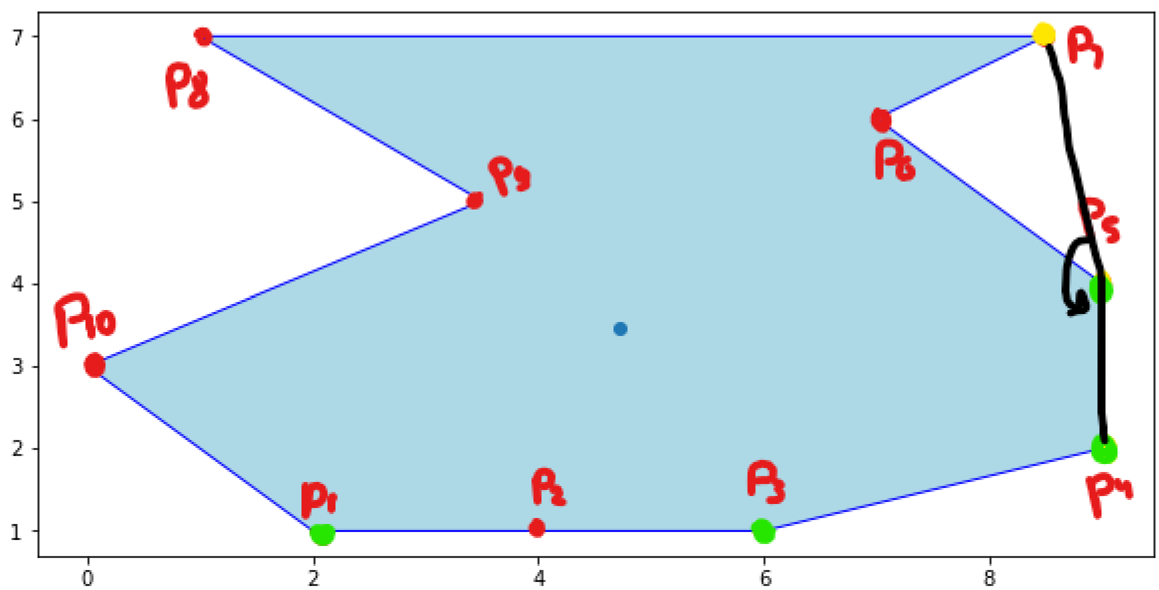
3. Tentamos inserir o próximo ponto da lista no fecho, p4 (em amarelo), e observamos que ele forma um ângulo counterclockwise com p1 e p3, então ele continua no fecho



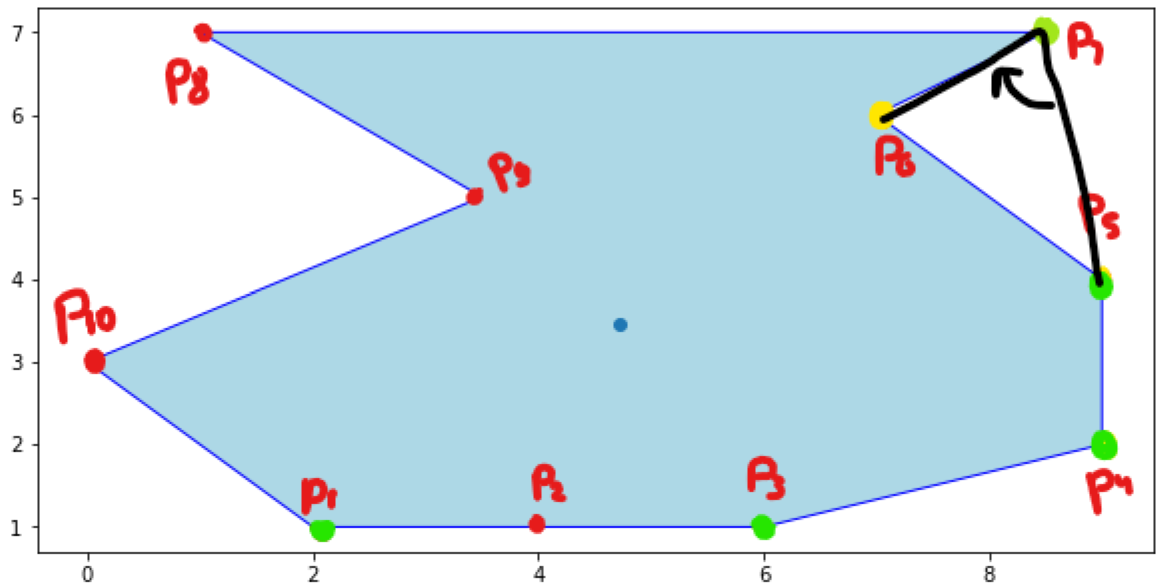
4. Testamos p5 com p4 e p3, temos um ângulo counterclockwise, adicionamos no fecho



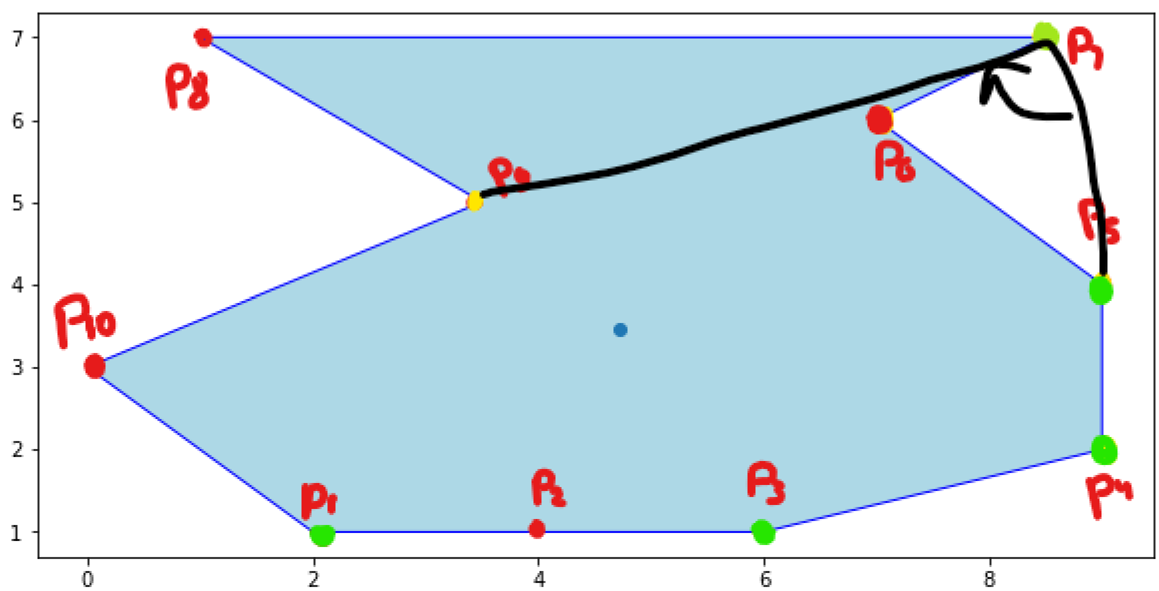
5. Testamos p7 com p5 e p4, ele é adicionado no fecho



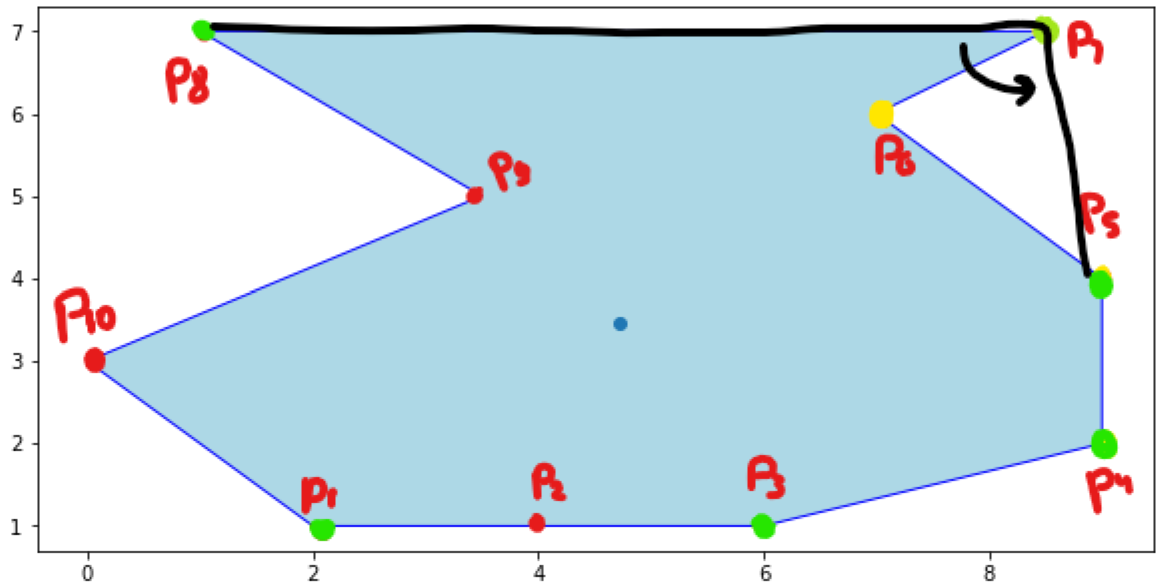
6. Testamos p6 com p7 e p5 e percebemos que o ângulo formado por eles não está na direção counter-clockwise, portanto p6 não está no fecho:



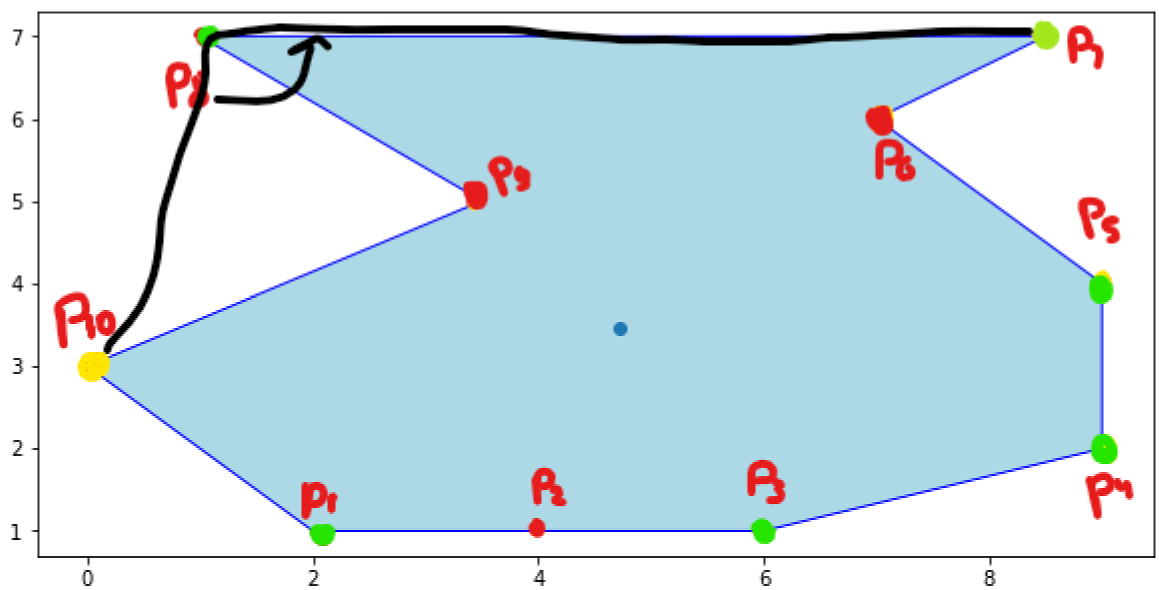
7. Testamos p9 com p7 e p5 e obtemos uma orientação clockwise, não pertence ao fecho:



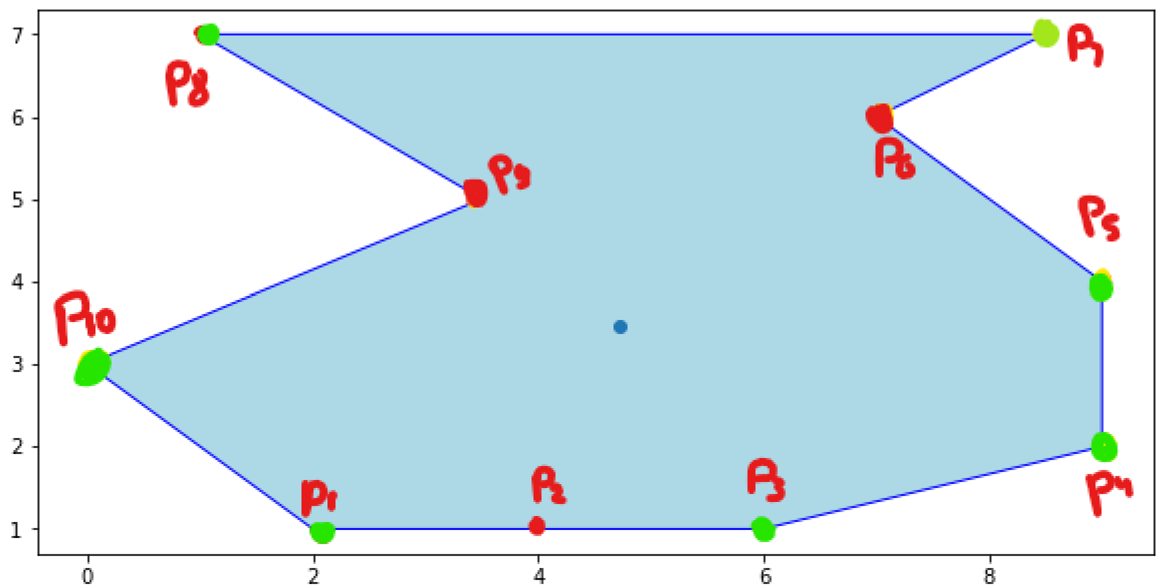
8. Testamos p8 com p7 e p5 e obtemos um ângulo counterclockwise, pertence ao fecho:



9. Testamos p10 com p8 e p7, pertence ao fecho:



10. Encerramos o fecho:



Item c

Porque precisamos de O(nlogn) para ordenar os pontos, e durante o loop principal, sempre que removemos um ponto do fecho não precisamos reexaminar todos os outros pontos para verificar se o fecho ainda é convexo, precisamos examinar apenas os pontos anteriores.

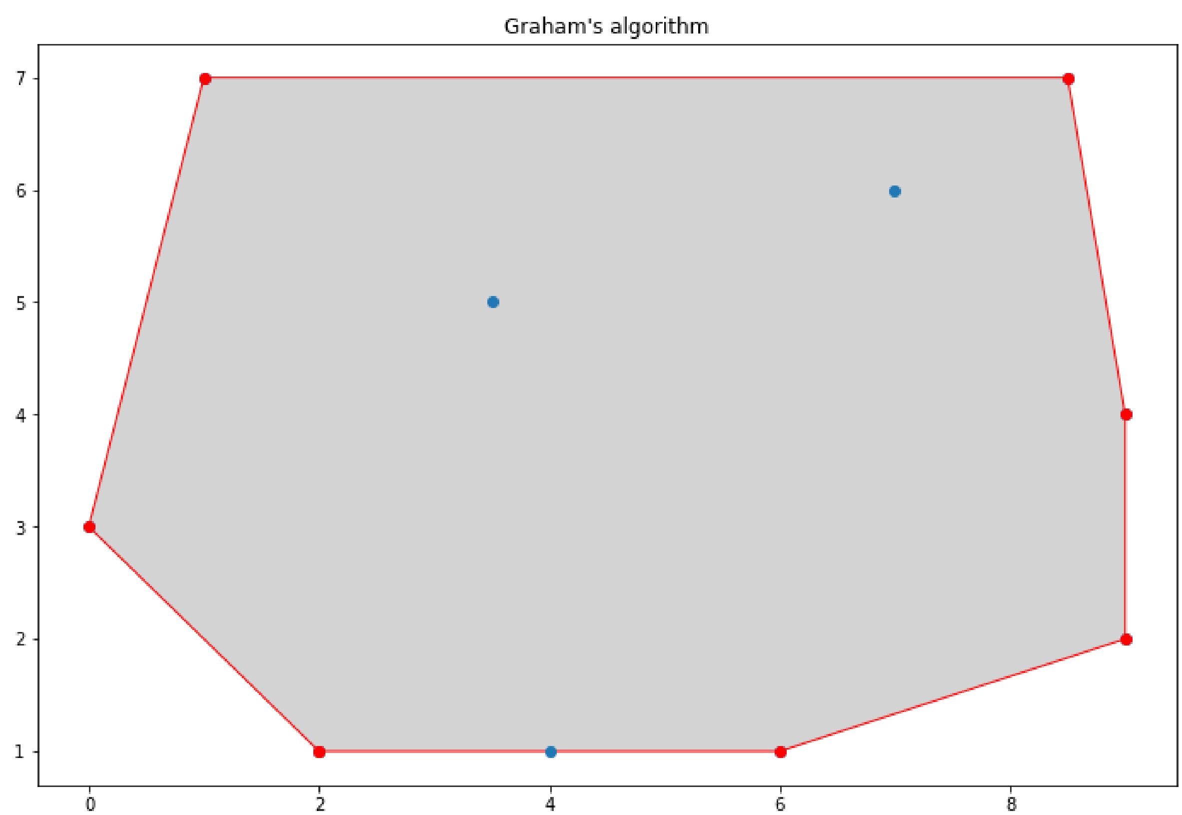
Item d

O código da implementação do algoritmo de Graham na linguagem Python é o seguinte:

```
def oriented_angle_from_zero(a,b):
    v = math.atan2(a[0]*b[1] - a[1]*b[0], a[0]*b[0] + a[1]*b[1])
    if math.isclose(v, 0):
        v = 0
    elif v < 0:
        v += math.pi * 2
    return v
def ccw(p1, p2, p3):
    z = (p2[0] - p1[0])*(p3[1] - p1[1]) - (p2[1] - p1[1])*(p3[0] - p1[0])
    return z

# Graham algorithm
def graham(P):
    # Sort by y-coord then x-coord
    P = P[P[:,0].argsort()]
    P = P[P[:,1].argsort(kind='mergesort')]
```

```
    # Get the first point
    p0 = P[0,:]

    # Order points by polar angle with p0, if multiple points have same polar angle then keep
only the farthest
    angles = {}
    for p in P:
        angle = round(oriented_angle_from_zero((1,0), p-p0), 2)
        if angle not in angles:
            angles[angle] = p
        elif np.linalg.norm(angles[angle]-p0) < np.linalg.norm(p-p0):
            angles[angle] = p

    sorted_keys = sorted(list(angles))
    sorted_values = [angles[k] for k in sorted_keys]

    stack = []
    for p in sorted_values:
        while len(stack) > 1 and ccw(stack[-2], stack[-1], p) <= 0:
            stack.pop()
        stack.append(p)
   stack.insert(0,p0)
    stack.append(p0)
    return stack
```

Aplicando ao polígono da questão, temos:

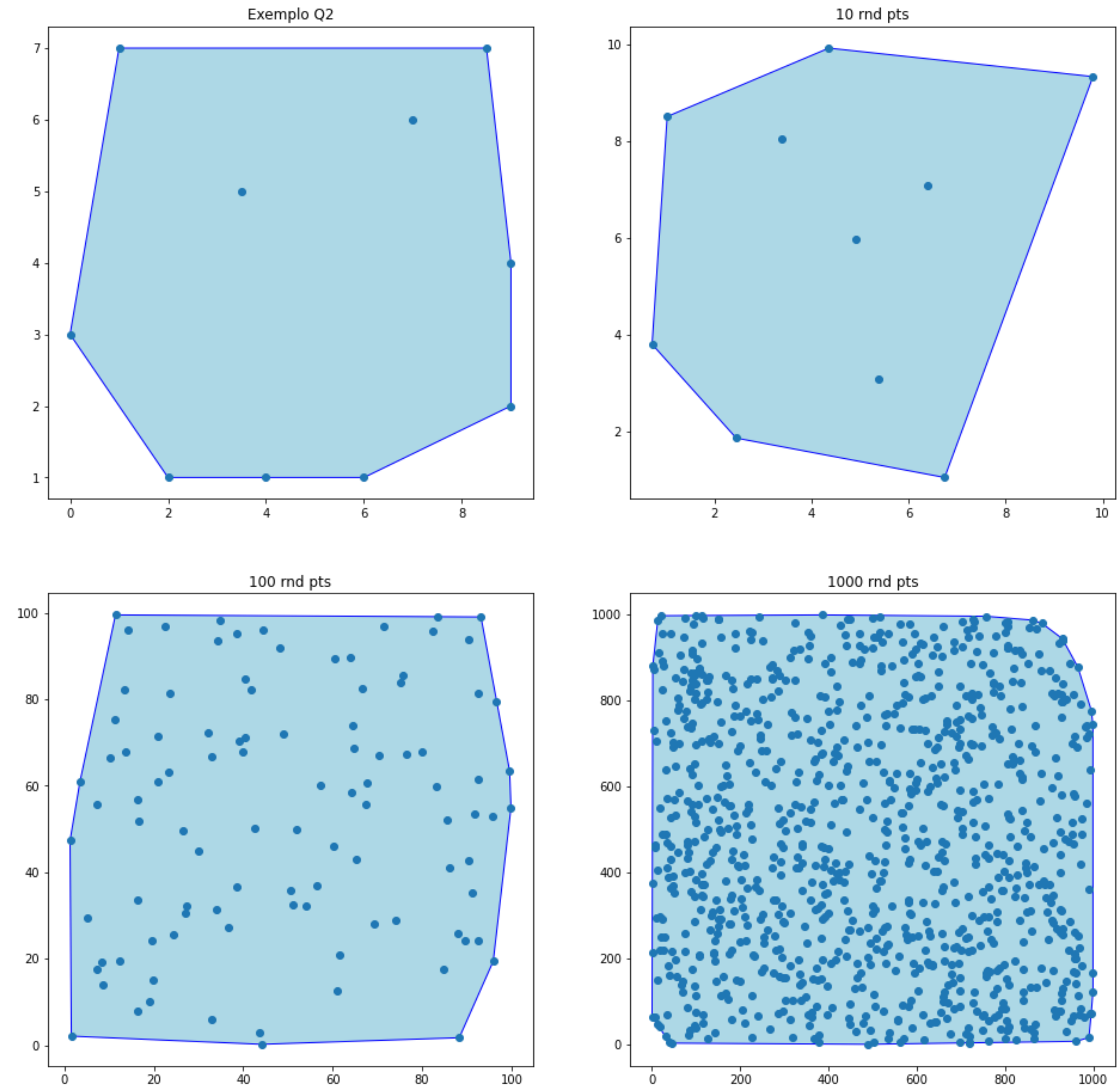


Questão 03

Implementação do algoritmo de Jarvis em Python:

```
# Jarvis algorithm
def prox(p0, v, P):
    min_angl = 1000000
    min_p = None
    min_i = -1
    for i in range(len(P)):
        p = P[i,:]
        if not np.allclose(p, p0):
            p0p = p - p0
            angle = oriented_angle_from_zero(v, p0p)
            if angle < min_angl:
                min_angl = angle
                min_p = p
                min_i = i
    return min_p, min_i


def jarvis(P):
    # Sort by y-coord then x-coord
    P = P[P[:,0].argsort()]
    P = P[P[:,1].argsort(kind='mergesort')]
    # Get the first point
    p0 = P[0,:]
    p1,i1 = prox(p0, (1,0), P)
    hull = [p0, p1]

    i = 1
    while not np.allclose(hull[i], p0):
        p_n, i_n = prox(hull[i], hull[i] - hull[i-1], P)
        hull.append(p_n)
        i += 1

    return hull
```

O resultado do algoritmo de Jarvis está nas figuras a seguir:



Comparando os tempos de Graham e Jarvis, temos:

	Tempo médio para 10 pontos (ms)	Tempo médio para 100 pontos (ms)	Tempo médio para 1000 pontos (ms)
Graham	0.0018	0.0080	0.1133
Jarvis	0.0173	0.2753	4.4345

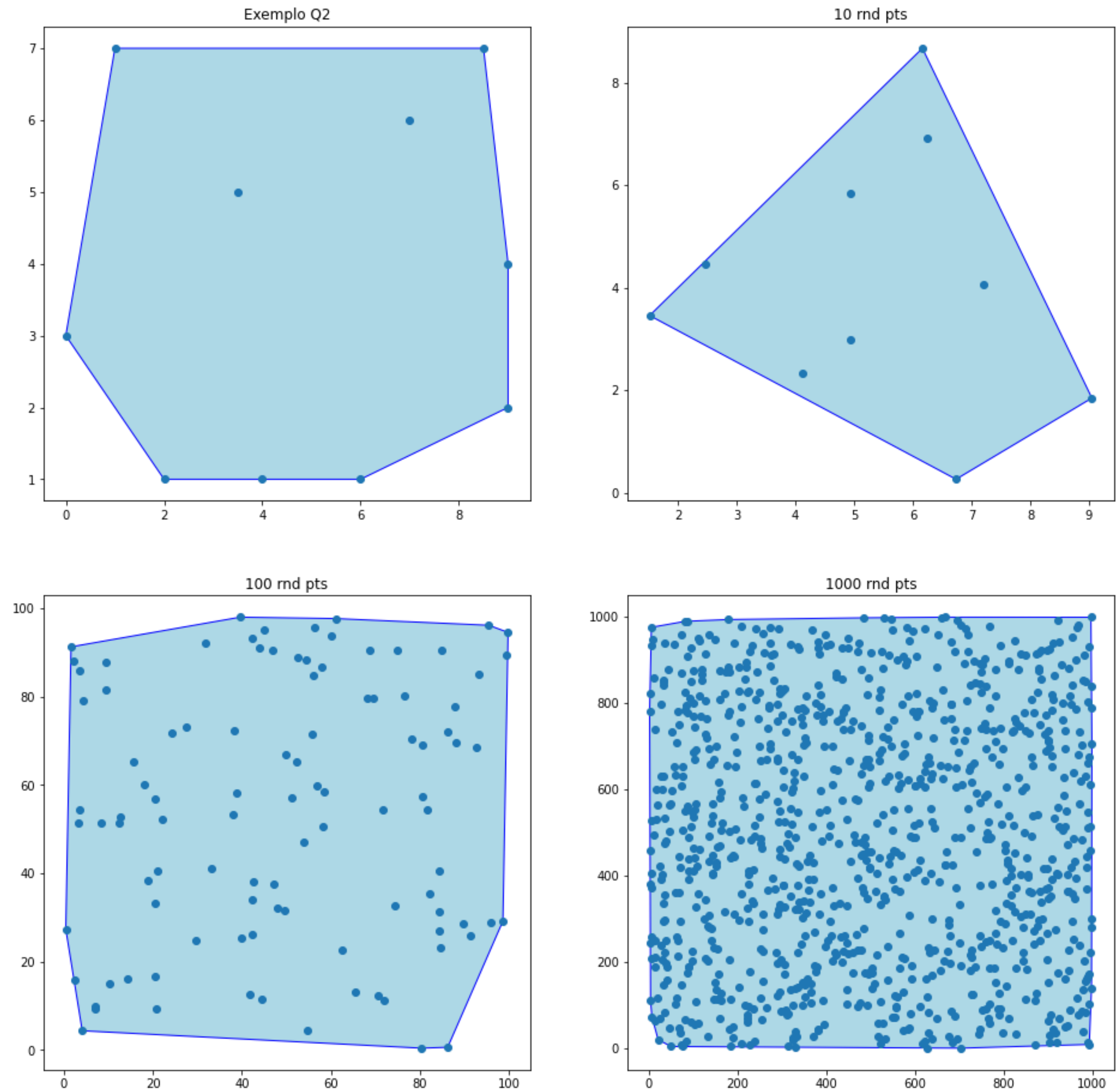
Observamos que o algoritmo de Jarvis, por ser $O(n^2)$, é mais lento que o de Graham, por ser O(nlogn).

Questão 04

Código do quickhull na linguagem Python:

```
def quickhull(P):
    hull = []
    # index of pts with min and max x coords
    min_x = 0
    max_x = 0

    for i in range(1,len(P)):
        p = P[i]
        if p[0] < P[min_x][0]:
            min_x = i
        elif p[0] == P[min_x][0] and p[1] < P[min_x][1]:
            min_x = i

        if p[0] > P[max_x][0]:
            max_x = i
        elif p[0] == P[max_x][0] and p[1] > P[max_x][1]:
            max_x = i

    # we have a line between P[min_x] and P[max_x] that divides the points
    # into S1 and S2
    pmin = P[min_x]
    pmax = P[max_x]

    hull.append(pmin)
    hull.append(pmax)

    S1 = []
    S2 = []
    for p in P:
        if not np.allclose(p, pmin) or np.allclose(p, pmax):
            d = direction(pmin, pmax, p)
            if d < 0:
                S1.append(p)
            elif d > 0:
                S2.append(p)

    findhull(S1, pmin, pmax, hull)
    findhull(S2, pmax, pmin, hull)

    return hull

def findhull(S, p, q, C):
    if len(S) == 0:
        return

    # Find farthest point on S from the segment pq
    max_dist = point_line_dist(p, q, S[0])
    max_i = 0
    for i in range(1, len(S)):
```

```
        dist = math.fabs(point_line_dist(p, q, S[i]))
        if dist > max_dist:
            max_dist = dist
            max_i = i

    # add point to hull between p and q
    pi = -1
    qi = -1
    for (i, c) in enumerate(C):
        if np.allclose(c, p):
            pi = i
        if np.allclose(c, q):
            qi = i

    c = S[max_i]
    C.insert(pi+1, c)

    S1 = []
    S2 = []
    for t in S:
        if not np.allclose(p, t) or np.allclose(q, t) or np.allclose(c, t):
            d = direction(p, c, t)
            if d < 0:
                S1.append(t)
            else:
                d2 = direction(c, q, t)
                if d2 < 0:
                    S2.append(t)

    findhull(S1, p, c, C)
    findhull(S2, c, q, C)
```



Analisando os tempos, temos:

	Tempo médio para 10 pontos (ms)	Tempo médio para 100 pontos (ms)	Tempo médio para 1000 pontos (ms)
Graham	0.0018	0.0080	0.1133
Jarvis	0.0173	0.2753	4.4345
Quickhull	0.0170	0.1442	1.0399

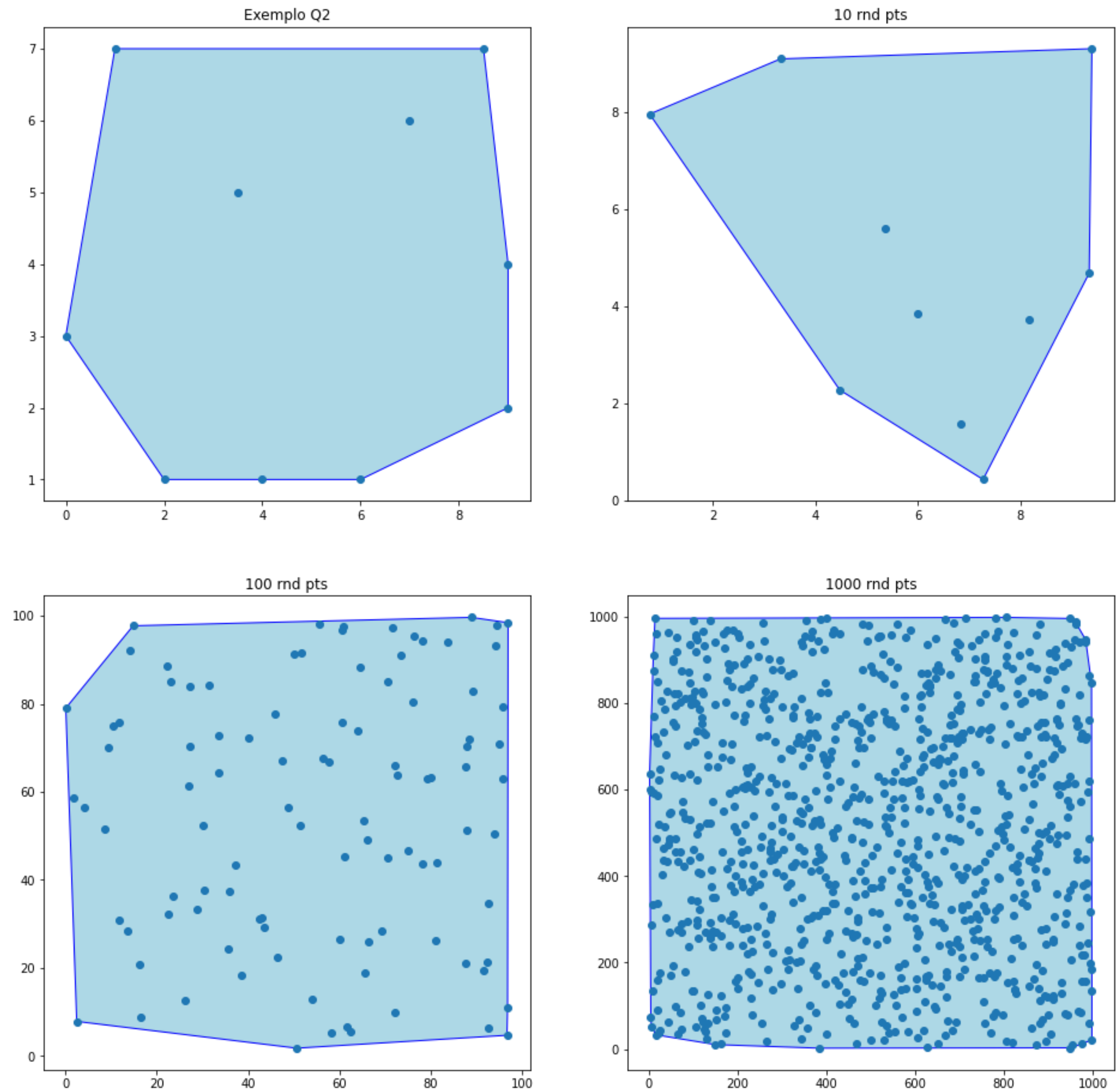
Questão 05

Código do mergehull na linguagem Python:

```
BASE_CASE_SIZE = 5

def get_next(i, L):
    if i == len(L)-1:
        return 0
    return i+1

def get_prev(i, L):
    if i == 0:
        return len(L)-1
    return i-1

# Receives two hulls in CCW order
def merge(left, right):
    # Remove the last point that closed the polygon
    left.pop()
    right.pop()
    hull = []
    left_i = 0 #index of leftmost point on left hull
    for i in range(1, len(left)):
        if left[i][0] > left[left_i][0]:
            left_i = i
    right_i = 0 #index of rightmost point on right hull
    for i in range(1, len(right)):
        if right[i][0] < right[right_i][0]:
            right_i = i

    p = left_i
    q = right_i

    prev_p = None
    prev_q = None

    # find upper tangent
    while True:
        prev_p = p
        prev_q = q
        while True:
            di = direction(right[q], left[p], left[get_next(p, left)])
            if di > 0: # continue while direction is not counter-clockwise
                break
            p = get_next(p, left)

        while True:
            di = direction(left[p], right[q], right[get_prev(q, right)])
            if di < 0: # continue while direction is not clockwise
                break
            q = get_prev(q, right)
```

```
        if p == prev_p and q == prev_q:
            break

    # find lower tangent
    cp = left_i
    cq = right_i

    while True:
        prev_p = cp
        prev_q = cq
        while True:
            di = direction(right[cq], left[cp], left[get_prev(cp, left)])
            if di < 0: # continue while direction is not clockwise
                break
            cp = get_prev(cp, left)

        while True:
            di = direction(left[cp], right[cq], right[get_next(cq, right)])
            if di > 0: # continue while direction is not counter-clockwise
                break
            cq = get_next(cq, right)

        if cp == prev_p and cq == prev_q:
            break

    #print(f'lower tangent is {left[cp]} {right[cq]}')

    # New hull is [p, cp] union [cq, q]
    if cp < p:
        hull.extend(left[p:])
        hull.extend(left[:cp+1])
    else:
        hull.extend(left[p:cp+1])

    if q < cq:
        hull.extend(right[cq:])
        hull.extend(right[:q+1])
    else:
        hull.extend(right[cq:q+1])

    # Close the hull
    hull.append(hull[0])

    return hull

def mergehull(P):
    # sort by x-coord
    P = P[P[:,0].argsort()]
    return mergehull_rec(P)

def mergehull_rec(P):
    n = len(P)
    # Base case
```

```
if n <= BASE_CASE_SIZE:
    hull = jarvis(P)
    return hull
mid = math.floor(n/2)
left_hull = mergehull_rec(P[:mid,:])
right_hull = mergehull_rec(P[mid:,:])
hull = merge(left_hull, right_hull)
return hull
```



Analisando os tempos, temos:

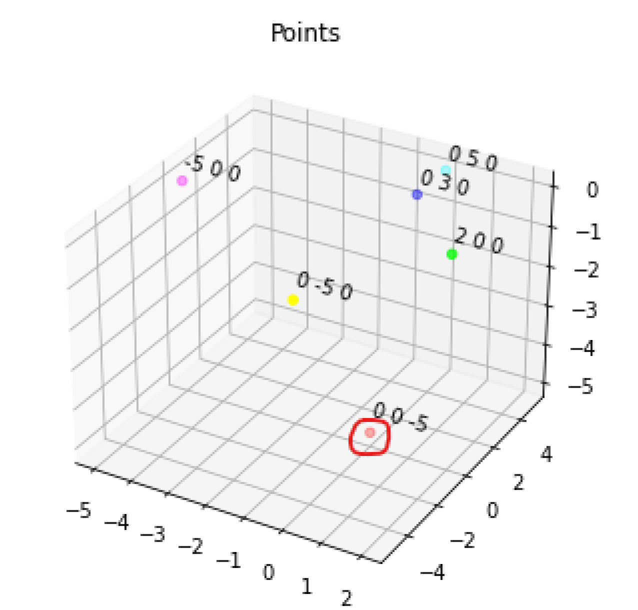
	Tempo médio para 10 pontos (ms)	Tempo médio para 100 pontos (ms)	Tempo médio para 1000 pontos (ms)

Graham	0.0018	0.0080	0.1133
Jarvis	0.0173	0.2753	4.4345
Quickhull	0.0170	0.1442	1.0399
Mergehull	0.0134	0.1089	1.2311

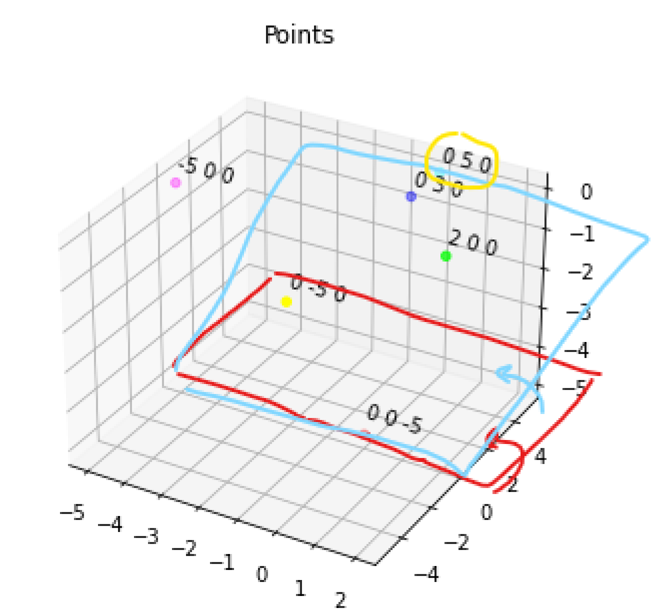
Questão 06

Item a

1. Começamos determinando o ponto de coordenada z mínima, (0,0,-5):

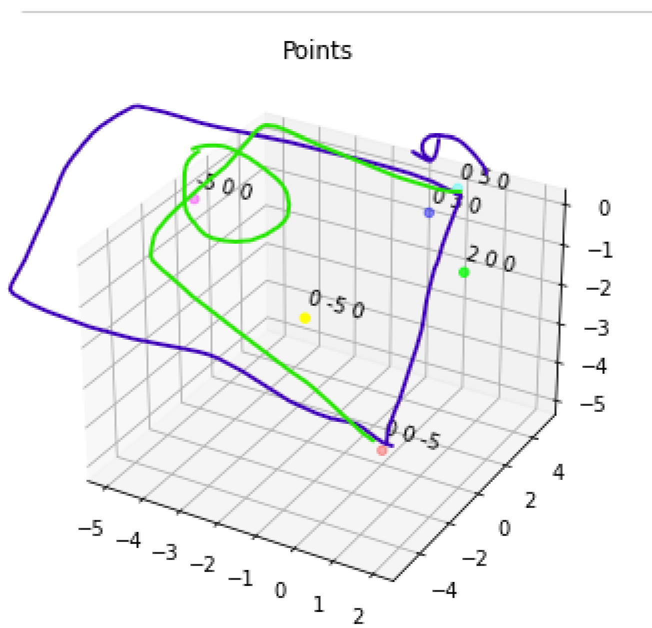


2. Formamos o plano com a reta suporte paralela ao eixo x e que contém o ponto mínimo. A reta é (0,0,-5)/(1,0,-5). Em seguida, rotacionamos esse plano até encontrar um ponto,

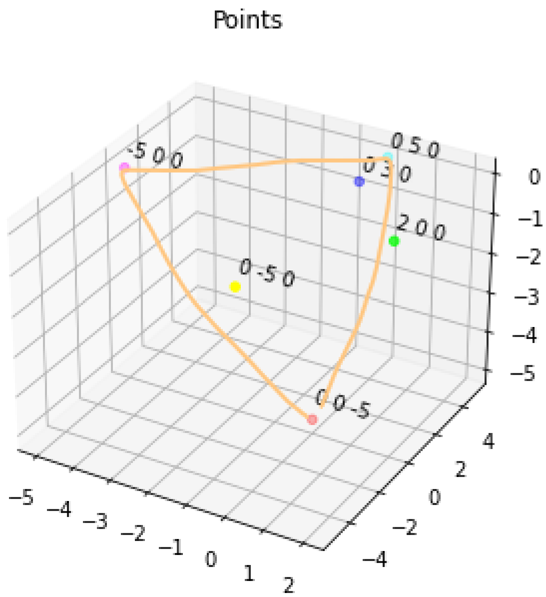


este é (0,5,0):

3. Formamos o plano com a reta suporte entre os pontos encontrados no passo 1 e 2, que são (0,0,-5) e (0,5,0). Ao girar o plano em torno da reta, obtemos o ponto (-5,0,0):



4. A face inicial é (0,0,-5), (0,5,0), (-5,0,0):



Item b/c

O código do algoritmo e sua execução são a seguinte:

```
def embrulho(F, p1, p2, P):
    min_cos = 10000000
    min_p = None
    n = np.cross(F[1]-F[0], F[2]-F[0])
    for p in P:
        if not np.allclose(p, F[0]) and not np.allclose(p, F[1]) and not np.allclose(p, F[2]):
            fp = np.cross(p-p1, p1-p2)
            cosp = np.dot(fp, n)/(np.linalg.norm(fp)*np.linalg.norm(n))
            if cosp < min_cos:
                min_cos = cosp
                min_p = p

    return min_p

from collections import deque

def get_face_exists(p1, p2, H):
    for f in H:
        if np.allclose(p1, f["points"][0]) and np.allclose(p2, f["points"][1]):
            return f, 0
        if np.allclose(p2, f["points"][0]) and np.allclose(p1, f["points"][1]):
            return f, 0
        if np.allclose(p1, f["points"][1]) and np.allclose(p2, f["points"][2]):
            return f, 1
```

```
        if np.allclose(p2, f["points"][1]) and np.allclose(p1, f["points"][2]):
            return f, 1
        if np.allclose(p1, f["points"][2]) and np.allclose(p2, f["points"][0]):
            return f, 2
        if np.allclose(p1, f["points"][2]) and np.allclose(p2, f["points"][0]):
            return f, 2
    return None, None


def embrulhohull(P, F):
    faces = deque([{"points": F, "livres": [0,1,2]}])
    hull = [faces[0]]

    while len(faces) > 0:
        work_face = faces.popleft()
        while len(work_face["livres"]) > 0:
            free_edge = work_face["livres"].pop()
            p1 = F[free_edge]
            p2 = F[get_next(free_edge, work_face["points"])]

            p3 = embrulho(work_face["points"], p1, p2, P)

            new_face = {"points": np.array([p1, p2, p3]), "livres": []}

            existing_face, existing_face_edge = get_face_exists(p2, p3, hull)
            if existing_face is None:
                new_face["livres"].append(1)
            else:
                if existing_face_edge in existing_face["livres"]:
                    existing_face["livres"].remove(existing_face_edge)

            existing_face, existing_face_edge = get_face_exists(p3, p1, hull)
            if existing_face is None:
                new_face["livres"].append(2)
            else:
                if existing_face_edge in existing_face["livres"]:
                    existing_face["livres"].remove(existing_face_edge)

            faces.append(new_face)
            hull.append(new_face)

    return hull
```

```
current working face: [[ 0  0 -5]
 [ 0  5  0]
 [-5  0  0]]
       look at free edge [-5  0  0] [ 0  0 -5]
              cosine with point [2 0 0] is 0.5773502691896258
              cosine with point [0 3 0] is 0.9684959969581862
              cosine with point [ 0 -5  0] is -0.3333333333333333
       point with min cosine is [ 0 -5  0]
       create face [[-5  0  0]
 [ 0  0 -5]
```

```
 [ 0 -5  0]]
       look at free edge [0 5 0] [-5  0  0]
              cosine with point [2 0 0] is 0.5773502691896258
              cosine with point [0 3 0] is 0.5773502691896258
              cosine with point [ 0 -5  0] is 0.5773502691896257
       point with min cosine is [ 0 -5  0]
       create face [[ 0  5  0]
 [-5  0  0]
 [ 0 -5  0]]
       look at free edge [ 0  0 -5] [0 5 0]
              cosine with point [2 0 0] is 0.10050378152592122
              cosine with point [0 3 0] is 0.5773502691896258
              cosine with point [ 0 -5  0] is 0.5773502691896257
       point with min cosine is [2 0 0]
       create face [[ 0  0 -5]
 [ 0  5  0]
 [ 2  0  0]]
current working face: [[-5  0  0]
 [ 0  0 -5]
 [ 0 -5  0]]
       look at free edge [-5  0  0] [ 0  0 -5]
              cosine with point [2 0 0] is 0.5773502691896258
              cosine with point [0 3 0] is -0.08804509063256238
              cosine with point [0 5 0] is -0.3333333333333333
       point with min cosine is [0 5 0]
       create face [[-5  0  0]
 [ 0  0 -5]
 [ 0  5  0]]
       look at free edge [0 5 0] [-5  0  0]
              cosine with point [2 0 0] is -0.5773502691896258
              cosine with point [0 3 0] is -0.5773502691896258
              cosine with point [0 5 0] is nan
       point with min cosine is [2 0 0]
       create face [[ 0  5  0]
 [-5  0  0]
 [ 2  0  0]]
current working face: [[ 0  5  0]
 [-5  0  0]
 [ 0 -5  0]]
       look at free edge [-5  0  0] [ 0  0 -5]
              cosine with point [ 0  0 -5] is nan
              cosine with point [2 0 0] is 0.0
              cosine with point [0 3 0] is 0.457495710997814
       point with min cosine is [2 0 0]
       create face [[-5  0  0]
 [ 0  0 -5]
 [ 2  0  0]]
       look at free edge [0 5 0] [-5  0  0]
              cosine with point [ 0  0 -5] is 0.5773502691896257
              cosine with point [2 0 0] is 1.0
              cosine with point [0 3 0] is 1.0
       point with min cosine is [ 0  0 -5]
       create face [[ 0  5  0]
```

```
 [-5  0  0]
 [ 0  0 -5]]
current working face: [[ 0  0 -5]
 [ 0  5  0]
 [ 2  0  0]]
       look at free edge [-5  0  0] [ 0  0 -5]
              cosine with point [0 3 0] is -0.026546593660094948
              cosine with point [ 0 -5  0] is -0.502518907629606
              cosine with point [-5  0  0] is nan
       point with min cosine is [ 0 -5  0]
       create face [[-5  0  0]
 [ 0  0 -5]
 [ 0 -5  0]]
current working face: [[-5  0  0]
 [ 0  0 -5]
 [ 0  5  0]]
current working face: [[ 0  5  0]
 [-5  0  0]
 [ 2  0  0]]
current working face: [[-5  0  0]
 [ 0  0 -5]
 [ 2  0  0]]
       look at free edge [0 5 0] [-5  0  0]
              cosine with point [0 3 0] is 0.0
              cosine with point [ 0 -5  0] is 0.0
              cosine with point [0 5 0] is nan
       point with min cosine is [0 3 0]
       create face [[ 0  5  0]
 [-5  0  0]
 [ 0  3  0]]
current working face: [[ 0  5  0]
 [-5  0  0]
 [ 0  0 -5]]
current working face: [[-5  0  0]
 [ 0  0 -5]
 [ 0 -5  0]]
current working face: [[ 0  5  0]
 [-5  0  0]
 [ 0  3  0]]
       look at free edge [-5  0  0] [ 0  0 -5]
              cosine with point [ 0  0 -5] is nan
              cosine with point [2 0 0] is 0.0
              cosine with point [ 0 -5  0] is -0.5773502691896258
       point with min cosine is [ 0 -5  0]
       create face [[-5  0  0]
 [ 0  0 -5]
 [ 0 -5  0]]
       look at free edge [0 5 0] [-5  0  0]
              cosine with point [ 0  0 -5] is 0.5773502691896258
              cosine with point [2 0 0] is 1.0
              cosine with point [ 0 -5  0] is 1.0
       point with min cosine is [ 0  0 -5]
       create face [[ 0  5  0]
```

```
 [-5  0  0]
 [ 0  0 -5]]
current working face: [[-5  0  0]
 [ 0  0 -5]
 [ 0 -5  0]]
current working face: [[ 0  5  0]
 [-5  0  0]
 [ 0  0 -5]]
```