# Geometria Computacional

[**Geometria Computacional**](#_udaenugahrb8)

**1**

[Calcular a distância entre dois pontos no plano.](#_wobnw6opduwn) 1

[**Encontrar o par de pontos mais próximo em um conjunto de pontos.**](#_x01c2nm3rc1) 2

[Verificar se um ponto está dentro de um polígono.](#_28a0v0sdrxtw) 3

[**Determinar se dois segmentos de reta se intersectam.**](#_ahrgb5h1lvnm) 5

[Encontrar o ponto de interseção entre dois segmentos de reta.](#_b5zbokd5i3ky) 6

[**Verificar se um ponto está dentro de um círculo.**](#_i1yqr4ju4eof) 8

[Calcular a interseção entre um círculo e um retângulo.](#_w809xpsx4xr) 9

[**Verificar se um polígono é convexo.**](#_mcctxnmm3h68) 10

[Determinar se duas linhas são paralelas ou concorrentes.](#_x13xd2nl89qs) 12

[**Encontrar o ponto de interseção entre duas linhas no espaço tridimensional.**](#_g95fox4t9x4n) 13

[Calcular o ângulo entre dois vetores.](#_e97mv0yy5ny9) 14

[**Verificar se três pontos estão em ordem anti-horária.**](#_e6i2bdqm4e3y) 15

## Calcular a distância entre dois pontos no plano.

Digite as coordenadas do primeiro ponto (x y): 5 6

Digite as coordenadas do segundo ponto (x y): 9 1

Distância entre os pontos: 6.4031242374328485

import math

def distance\_between\_points(x1, y1, x2, y2):

return math.sqrt((x2 - x1)\*\*2 + (y2 - y1)\*\*2)

# Solicita a entrada do usuário para as coordenadas dos pontos

x1, y1 = map(float, input("Digite as coordenadas do primeiro ponto (x y): ").split())

x2, y2 = map(float, input("Digite as coordenadas do segundo ponto (x y): ").split())

# Chama a função para calcular a distância entre os pontos

distancia = distance\_between\_points(x1, y1, x2, y2)

# Exibe a distância entre os pontos

print("Distância entre os pontos:", distancia)

## Encontrar o par de pontos mais próximo em um conjunto de pontos.

Digite o número de pontos: 4

Digite as coordenadas dos pontos (x y) separados por espaço:

1 6

9 7

5 6

3 5

Distância entre os pontos mais próximos: 2.23606797749979

Par de pontos mais próximo: {'x': 1.0, 'y': 6.0} {'x': 3.0, 'y': 5.0}

import math

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(p1, p2):

return math.sqrt((p1.x - p2.x)\*\*2 + (p1.y - p2.y)\*\*2)

def brute\_force\_closest\_pair(points, left, right):

min\_distance = float('inf')

closest\_pair = None

for i in range(left, right):

for j in range(i + 1, right):

dist = distance(points[i], points[j])

if dist < min\_distance:

min\_distance = dist

closest\_pair = (points[i], points[j])

return min\_distance, closest\_pair

def closest\_pair(points, left, right):

if right - left <= 3:

return brute\_force\_closest\_pair(points, left, right)

mid = (left + right) // 2

dl, pair\_left = closest\_pair(points, left, mid)

dr, pair\_right = closest\_pair(points, mid, right)

min\_dist = min(dl, dr)

best\_pair = pair\_left if dl < dr else pair\_right

strip = []

for i in range(left, right):

if abs(points[i].x - points[mid].x) < min\_dist:

strip.append(points[i])

strip.sort(key=lambda point: point.y)

for i in range(len(strip)):

for j in range(i + 1, min(i + 7, len(strip))):

dist = distance(strip[i], strip[j])

if dist < min\_dist:

min\_dist = dist

best\_pair = (strip[i], strip[j])

return min\_dist, best\_pair

def closest\_points(points):

points.sort(key=lambda point: point.x)

return closest\_pair(points, 0, len(points))

# Solicita a entrada do usuário para o número de pontos

num\_points = int(input("Digite o número de pontos: "))

points = []

print("Digite as coordenadas dos pontos (x y) separados por espaço:")

for \_ in range(num\_points):

x, y = map(float, input().split())

points.append(Point(x, y))

# Chama a função para encontrar o par de pontos mais próximo

min\_distance, closest\_pair = closest\_points(points)

# Exibe a distância e os pontos do par mais próximo

print("Distância entre os pontos mais próximos:", min\_distance)

print("Par de pontos mais próximo:", closest\_pair[0].\_\_dict\_\_, closest\_pair[1].\_\_dict\_\_)

## Verificar se um ponto está dentro de um polígono.

Digite o número de vértices do polígono: 4

Digite as coordenadas dos vértices do polígono (x y) separados por espaço:

5 9

6 2

1 8

3 4

Digite as coordenadas do ponto (x y) a verificar: 1 8

O ponto está dentro do polígono.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def orientation(p, q, r):

val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)

if val == 0:

return 0 # Colinear

return 1 if val > 0 else 2 # Sentido horário ou anti-horário

def is\_inside\_polygon(polygon, n, point):

if n < 3:

return False

extreme = Point(float("inf"), point.y)

count = 0

i = 0

while True:

next\_idx = (i + 1) % n

if (orientation(polygon[i], polygon[next\_idx], point) == 0 and

point.x >= min(polygon[i].x, polygon[next\_idx].x) and

point.x <= max(polygon[i].x, polygon[next\_idx].x)):

return True

if (orientation(polygon[i], point, extreme) == 0 and

point.x >= min(polygon[i].x, extreme.x) and

point.x <= max(polygon[i].x, extreme.x)):

return True

if (polygon[i].y > point.y) != (polygon[next\_idx].y > point.y):

if point.x <= (polygon[i].x + (point.y - polygon[i].y) \* (polygon[next\_idx].x - polygon[i].x) / (polygon[next\_idx].y - polygon[i].y)):

count += 1

i = next\_idx

if i == 0:

break

return count % 2 == 1

# Solicita a entrada do usuário para as coordenadas dos vértices do polígono

num\_vertices = int(input("Digite o número de vértices do polígono: "))

polygon = []

print("Digite as coordenadas dos vértices do polígono (x y) separados por espaço:")

for \_ in range(num\_vertices):

x, y = map(float, input().split())

polygon.append(Point(x, y))

# Solicita a entrada do usuário para as coordenadas do ponto a verificar

x\_point, y\_point = map(float, input("Digite as coordenadas do ponto (x y) a verificar: ").split())

point\_to\_check = Point(x\_point, y\_point)

# Verifica se o ponto está dentro do polígono

inside = is\_inside\_polygon(polygon, num\_vertices, point\_to\_check)

# Exibe o resultado

if inside:

print("O ponto está dentro do polígono.")

else:

print("O ponto não está dentro do polígono.")

## Determinar se dois segmentos de reta se intersectam.

Digite as coordenadas do primeiro segmento (x1 y1 x2 y2): 1 8 6 4

Digite as coordenadas do segundo segmento (x1 y1 x2 y2): 2 8 7 3

Os segmentos de reta se intersectam.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def orientation(p, q, r):

val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)

if val == 0:

return 0 # Colinear

return 1 if val > 0 else 2 # Sentido horário ou anti-horário

def on\_segment(p, q, r):

return (q.x <= max(p.x, r.x) and q.x >= min(p.x, r.x) and

q.y <= max(p.y, r.y) and q.y >= min(p.y, r.y))

def do\_segments\_intersect(p1, q1, p2, q2):

o1 = orientation(p1, q1, p2)

o2 = orientation(p1, q1, q2)

o3 = orientation(p2, q2, p1)

o4 = orientation(p2, q2, q1)

if o1 != o2 and o3 != o4:

return True

if o1 == 0 and on\_segment(p1, p2, q1):

return True

if o2 == 0 and on\_segment(p1, q2, q1):

return True

if o3 == 0 and on\_segment(p2, p1, q2):

return True

if o4 == 0 and on\_segment(p2, q1, q2):

return True

return False

# Solicita a entrada do usuário para as coordenadas dos extremos dos segmentos

x1, y1, x2, y2 = map(float, input("Digite as coordenadas do primeiro segmento (x1 y1 x2 y2): ").split())

x3, y3, x4, y4 = map(float, input("Digite as coordenadas do segundo segmento (x1 y1 x2 y2): ").split())

p1 = Point(x1, y1)

q1 = Point(x2, y2)

p2 = Point(x3, y3)

q2 = Point(x4, y4)

# Verifica se os segmentos se intersectam

if do\_segments\_intersect(p1, q1, p2, q2):

print("Os segmentos de reta se intersectam.")

else:

print("Os segmentos de reta não se intersectam.")

## Encontrar o ponto de interseção entre dois segmentos de reta.

Digite as coordenadas do primeiro segmento (x1 y1 x2 y2): 5 9 8 6

Digite as coordenadas do segundo segmento (x1 y1 x2 y2): 2 4 8 6

Ponto de interseção: 8.0 6.0

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def orientation(p, q, r):

val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)

if val == 0:

return 0 # Colinear

return 1 if val > 0 else 2 # Sentido horário ou anti-horário

def on\_segment(p, q, r):

return (q.x <= max(p.x, r.x) and q.x >= min(p.x, r.x) and

q.y <= max(p.y, r.y) and q.y >= min(p.y, r.y))

def do\_segments\_intersect(p1, q1, p2, q2):

o1 = orientation(p1, q1, p2)

o2 = orientation(p1, q1, q2)

o3 = orientation(p2, q2, p1)

o4 = orientation(p2, q2, q1)

if o1 != o2 and o3 != o4:

return True

if o1 == 0 and on\_segment(p1, p2, q1):

return True

if o2 == 0 and on\_segment(p1, q2, q1):

return True

if o3 == 0 and on\_segment(p2, p1, q2):

return True

if o4 == 0 and on\_segment(p2, q1, q2):

return True

return False

def intersection\_point(p1, q1, p2, q2):

if not do\_segments\_intersect(p1, q1, p2, q2):

return None

A1 = q1.y - p1.y

B1 = p1.x - q1.x

C1 = A1 \* p1.x + B1 \* p1.y

A2 = q2.y - p2.y

B2 = p2.x - q2.x

C2 = A2 \* p2.x + B2 \* p2.y

det = A1 \* B2 - A2 \* B1

x = (B2 \* C1 - B1 \* C2) / det

y = (A1 \* C2 - A2 \* C1) / det

return Point(x, y)

# Solicita a entrada do usuário para as coordenadas dos extremos dos segmentos

x1, y1, x2, y2 = map(float, input("Digite as coordenadas do primeiro segmento (x1 y1 x2 y2): ").split())

x3, y3, x4, y4 = map(float, input("Digite as coordenadas do segundo segmento (x1 y1 x2 y2): ").split())

p1 = Point(x1, y1)

q1 = Point(x2, y2)

p2 = Point(x3, y3)

q2 = Point(x4, y4)

# Encontra o ponto de interseção entre os segmentos

intersection = intersection\_point(p1, q1, p2, q2)

# Exibe o ponto de interseção, se existir

if intersection:

print("Ponto de interseção:", intersection.x, intersection.y)

else:

print("Os segmentos não se intersectam.")

## Verificar se um ponto está dentro de um círculo.

Digite as coordenadas do centro do círculo (x y): 5 9

Digite o raio do círculo: 2

Digite as coordenadas do ponto (x y) a verificar: 1 8

O ponto não está dentro do círculo.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(p1, p2):

return ((p2.x - p1.x)\*\*2 + (p2.y - p1.y)\*\*2)\*\*0.5

def is\_inside\_circle(center, radius, point):

dist = distance(center, point)

return dist <= radius

# Solicita a entrada do usuário para as coordenadas do centro do círculo

center\_x, center\_y = map(float, input("Digite as coordenadas do centro do círculo (x y): ").split())

center = Point(center\_x, center\_y)

radius = float(input("Digite o raio do círculo: "))

# Solicita a entrada do usuário para as coordenadas do ponto a verificar

point\_x, point\_y = map(float, input("Digite as coordenadas do ponto (x y) a verificar: ").split())

point = Point(point\_x, point\_y)

# Verifica se o ponto está dentro do círculo

if is\_inside\_circle(center, radius, point):

print("O ponto está dentro do círculo.")

else:

print("O ponto não está dentro do círculo.")

## Calcular a interseção entre um círculo e um retângulo.

Digite as coordenadas do centro do círculo (x y): 5 9

Digite o raio do círculo: 6

Digite as coordenadas dos vértices do retângulo (x1 y1 x2 y2):

4 8 9 5

O círculo tem interseção com o retângulo.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(p1, p2):

return ((p2.x - p1.x)\*\*2 + (p2.y - p1.y)\*\*2)\*\*0.5

def is\_inside\_rectangle(rectangle, point):

return (rectangle[0].x <= point.x <= rectangle[1].x and

rectangle[0].y <= point.y <= rectangle[1].y)

def circle\_rectangle\_intersection(center, radius, rectangle):

closest\_x = max(rectangle[0].x, min(center.x, rectangle[1].x))

closest\_y = max(rectangle[0].y, min(center.y, rectangle[1].y))

closest\_point = Point(closest\_x, closest\_y)

if distance(center, closest\_point) <= radius:

return True

return False

# Solicita a entrada do usuário para as coordenadas do centro do círculo

center\_x, center\_y = map(float, input("Digite as coordenadas do centro do círculo (x y): ").split())

center = Point(center\_x, center\_y)

radius = float(input("Digite o raio do círculo: "))

# Solicita a entrada do usuário para as coordenadas dos vértices do retângulo

print("Digite as coordenadas dos vértices do retângulo (x1 y1 x2 y2):")

rectangle\_x1, rectangle\_y1, rectangle\_x2, rectangle\_y2 = map(float, input().split())

rectangle = [Point(rectangle\_x1, rectangle\_y1), Point(rectangle\_x2, rectangle\_y2)]

# Verifica a interseção entre o círculo e o retângulo

if is\_inside\_rectangle(rectangle, center):

print("O círculo está totalmente dentro do retângulo.")

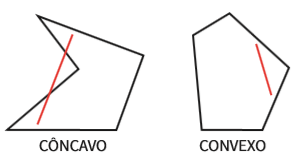
elif circle\_rectangle\_intersection(center, radius, rectangle):

print("O círculo tem interseção com o retângulo.")

else:

print("O círculo e o retângulo não se intersectam.")

## Verificar se um polígono é convexo.



Digite o número de vértices do polígono: 5

Digite as coordenadas dos vértices do polígono (x y) separados por espaço:

10 5

2 9

6 7

3 4

1 11

O polígono é convexo.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def orientation(p, q, r):

val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)

if val == 0:

return 0 # Colinear

return 1 if val > 0 else 2 # Sentido horário ou anti-horário

def is\_convex\_polygon(vertices):

n = len(vertices)

if n < 3:

return False

direction = 0

for i in range(n):

p = vertices[i]

q = vertices[(i + 1) % n]

r = vertices[(i + 2) % n]

o = orientation(p, q, r)

if o == 0:

continue

if direction == 0:

direction = o

elif direction != o:

return False

return True

# Solicita a entrada do usuário para o número de vértices do polígono

num\_vertices = int(input("Digite o número de vértices do polígono: "))

polygon = []

print("Digite as coordenadas dos vértices do polígono (x y) separados por espaço:")

for \_ in range(num\_vertices):

x, y = map(float, input().split())

polygon.append(Point(x, y))

# Verifica se o polígono é convexo

if is\_convex\_polygon(polygon):

print("O polígono é convexo.")

else:

print("O polígono não é convexo.")

## Determinar se duas linhas são paralelas ou concorrentes.

Digite as coordenadas do primeiro ponto da primeira linha (x y): 11 1

Digite as coordenadas do segundo ponto da primeira linha (x y): 5 9

Digite as coordenadas do primeiro ponto da segunda linha (x y): 3 7

Digite as coordenadas do segundo ponto da segunda linha (x y): 20 8

As linhas são concorrentes e se intersectam no ponto: 6.352112676056338 7.197183098591549

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

class Line:

def \_\_init\_\_(self, point1, point2):

self.point1 = point1

self.point2 = point2

def slope(line):

return (line.point2.y - line.point1.y) / (line.point2.x - line.point1.x)

def are\_parallel(line1, line2):

return slope(line1) == slope(line2)

def intersection\_point(line1, line2):

x1, y1 = line1.point1.x, line1.point1.y

x2, y2 = line1.point2.x, line1.point2.y

x3, y3 = line2.point1.x, line2.point1.y

x4, y4 = line2.point2.x, line2.point2.y

det = (x1 - x2) \* (y3 - y4) - (y1 - y2) \* (x3 - x4)

if det == 0:

return None # Linhas paralelas ou coincidentes

px = ((x1 \* y2 - y1 \* x2) \* (x3 - x4) - (x1 - x2) \* (x3 \* y4 - y3 \* x4)) / det

py = ((x1 \* y2 - y1 \* x2) \* (y3 - y4) - (y1 - y2) \* (x3 \* y4 - y3 \* x4)) / det

return Point(px, py)

# Solicita a entrada do usuário para as coordenadas dos pontos das duas linhas

x1, y1 = map(float, input("Digite as coordenadas do primeiro ponto da primeira linha (x y): ").split())

x2, y2 = map(float, input("Digite as coordenadas do segundo ponto da primeira linha (x y): ").split())

line1 = Line(Point(x1, y1), Point(x2, y2))

x3, y3 = map(float, input("Digite as coordenadas do primeiro ponto da segunda linha (x y): ").split())

x4, y4 = map(float, input("Digite as coordenadas do segundo ponto da segunda linha (x y): ").split())

line2 = Line(Point(x3, y3), Point(x4, y4))

# Verifica se as linhas são paralelas ou concorrentes

if are\_parallel(line1, line2):

print("As linhas são paralelas.")

else:

intersection = intersection\_point(line1, line2)

if intersection:

print("As linhas são concorrentes e se intersectam no ponto:", intersection.x, intersection.y)

else:

print("As linhas são concorrentes, mas não se intersectam.")

## Encontrar o ponto de interseção entre duas linhas no espaço tridimensional.

Digite as coordenadas do ponto inicial da primeira linha (x y z): 2 6 8

Digite as coordenadas do vetor diretor da primeira linha (a b c): 1 5 9

Digite as coordenadas do ponto inicial da segunda linha (x y z): 3 4 8

Digite as coordenadas do vetor diretor da segunda linha (a b c): 2 0 7

O ponto de interseção é: 3.0 11.0 17.0

class Point3D:

def \_\_init\_\_(self, x, y, z):

self.x = x

self.y = y

self.z = z

class Line3D:

def \_\_init\_\_(self, point, direction):

self.point = point

self.direction = direction

def find\_intersection(line1, line2):

# Calculate direction ratios for both lines

a1, b1, c1 = line1.direction

a2, b2, c2 = line2.direction

# Solve linear equations to find parameters t1 and t2

t1 = (line2.point.x - line1.point.x) / a1

t2 = ((line1.point.x - line2.point.x) \* b2 - (line1.point.y - line2.point.y) \* a2) / (a1 \* b2 - a2 \* b1)

# Calculate intersection point coordinates

x = line1.point.x + t1 \* a1

y = line1.point.y + t1 \* b1

z = line1.point.z + t1 \* c1

return Point3D(x, y, z)

# Solicita a entrada do usuário para as coordenadas do ponto inicial e do vetor diretor da primeira linha

x1, y1, z1 = map(float, input("Digite as coordenadas do ponto inicial da primeira linha (x y z): ").split())

a1, b1, c1 = map(float, input("Digite as coordenadas do vetor diretor da primeira linha (a b c): ").split())

line1 = Line3D(Point3D(x1, y1, z1), (a1, b1, c1))

# Solicita a entrada do usuário para as coordenadas do ponto inicial e do vetor diretor da segunda linha

x2, y2, z2 = map(float, input("Digite as coordenadas do ponto inicial da segunda linha (x y z): ").split())

a2, b2, c2 = map(float, input("Digite as coordenadas do vetor diretor da segunda linha (a b c): ").split())

line2 = Line3D(Point3D(x2, y2, z2), (a2, b2, c2))

# Encontra o ponto de interseção entre as linhas

intersection = find\_intersection(line1, line2)

print("O ponto de interseção é:", intersection.x, intersection.y, intersection.z)

## Calcular o ângulo entre dois vetores.

Digite as coordenadas do primeiro vetor (x y z): 2 5 6

Digite as coordenadas do segundo vetor (x y z): 3 4 8

O ângulo entre os vetores é: 13.362707311080856 graus

import math

class Vector:

def \_\_init\_\_(self, x, y, z):

self.x = x

self.y = y

self.z = z

def dot\_product(vector1, vector2):

return vector1.x \* vector2.x + vector1.y \* vector2.y + vector1.z \* vector2.z

def vector\_magnitude(vector):

return math.sqrt(vector.x\*\*2 + vector.y\*\*2 + vector.z\*\*2)

def angle\_between\_vectors(vector1, vector2):

dot = dot\_product(vector1, vector2)

magnitude1 = vector\_magnitude(vector1)

magnitude2 = vector\_magnitude(vector2)

cosine\_theta = dot / (magnitude1 \* magnitude2)

radians = math.acos(cosine\_theta)

# Converte de radianos para graus

degrees = math.degrees(radians)

return degrees

# Solicita a entrada do usuário para as coordenadas dos vetores

x1, y1, z1 = map(float, input("Digite as coordenadas do primeiro vetor (x y z): ").split())

vector1 = Vector(x1, y1, z1)

x2, y2, z2 = map(float, input("Digite as coordenadas do segundo vetor (x y z): ").split())

vector2 = Vector(x2, y2, z2)

# Calcula o ângulo entre os vetores

angle = angle\_between\_vectors(vector1, vector2)

print("O ângulo entre os vetores é:", angle, "graus")

## Verificar se três pontos estão em ordem anti-horária.

Digite as coordenadas do primeiro ponto (x y): 1 9

Digite as coordenadas do segundo ponto (x y): 3 5

Digite as coordenadas do terceiro ponto (x y): 2 7

Digite as coordenadas do ponto de referência (x y): 1 6

Os três pontos não estão em ordem anti-horária em relação ao ponto de referência.

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def cross\_product(point1, point2, point\_ref):

return (point2.x - point1.x) \* (point\_ref.y - point1.y) - (point2.y - point1.y) \* (point\_ref.x - point1.x)

def check\_counterclockwise(point1, point2, point3):

return cross\_product(point1, point2, point3) > 0

# Solicita a entrada do usuário para as coordenadas dos pontos

x1, y1 = map(float, input("Digite as coordenadas do primeiro ponto (x y): ").split())

x2, y2 = map(float, input("Digite as coordenadas do segundo ponto (x y): ").split())

x3, y3 = map(float, input("Digite as coordenadas do terceiro ponto (x y): ").split())

x\_ref, y\_ref = map(float, input("Digite as coordenadas do ponto de referência (x y): ").split())

point1 = Point(x1, y1)

point2 = Point(x2, y2)

point3 = Point(x3, y3)

point\_ref = Point(x\_ref, y\_ref)

# Verifica se os três pontos estão em ordem anti-horária

if check\_counterclockwise(point1, point2, point3):

print("Os três pontos estão em ordem anti-horária em relação ao ponto de referência.")

else:

print("Os três pontos não estão em ordem anti-horária em relação ao ponto de referência.")