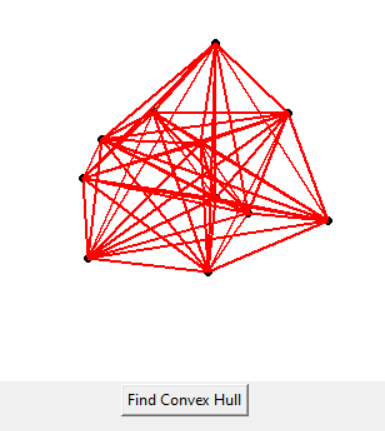
BRUTE FORCE

import tkinter as tk  
from itertools import combinations  
  
class ConvexHullApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Convex Hull Finder")  
  
 self.canvas = tk.Canvas(self.master, width=400, height=400, bg="white")  
 self.canvas.pack()  
  
 self.points = []  
  
 self.canvas.bind("<Button-1>", self.on\_canvas\_click)  
  
 self.find\_hull\_button = tk.Button(self.master, text="Find Convex Hull", command=self.find\_convex\_hull)  
 self.find\_hull\_button.pack()  
  
 def on\_canvas\_click(self, event):  
 x, y = event.x, event.y  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="black")  
 self.points.append((x, y))  
  
 def find\_convex\_hull(self):  
 convex\_hull = self.brute\_force\_convex\_hull(self.points)  
  
 # Clear previous hull if any  
 self.canvas.delete("hull")  
  
 # Draw convex hull  
 for i in range(len(convex\_hull)):  
 p1 = convex\_hull[i]  
 p2 = convex\_hull[(i + 1) % len(convex\_hull)]  
 self.canvas.create\_line(p1, p2, fill="red", width=2, tags="hull")  
  
 def orientation(self, p, q, r):  
 val = (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1])  
 if val == 0:  
 return 0  
 return 1 if val > 0 else -1  
  
 def is\_inside(self, points, p):  
 n = len(points)  
 if n < 3:  
 return False  
  
 for i in range(n):  
 if self.orientation(points[i], points[(i + 1) % n], p) != -1:  
 return False  
  
 return True  
  
 def brute\_force\_convex\_hull(self, points):  
 n = len(points)  
 convex\_hull = []  
  
 for subset in combinations(points, 3):  
 subset = list(subset)  
 p1, p2, p3 = subset  
 if self.orientation(p1, p2, p3) != 0:  
 if self.is\_inside(convex\_hull, p1):  
 convex\_hull.remove(p1)  
 convex\_hull.append(p1)  
 if self.is\_inside(convex\_hull, p2):  
 convex\_hull.remove(p2)  
 convex\_hull.append(p2)  
 if self.is\_inside(convex\_hull, p3):  
 convex\_hull.remove(p3)  
 convex\_hull.append(p3)  
  
 return convex\_hull  
  
  
def main():  
 root = tk.Tk()  
 app = ConvexHullApp(root)  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



JARVIS MARCH

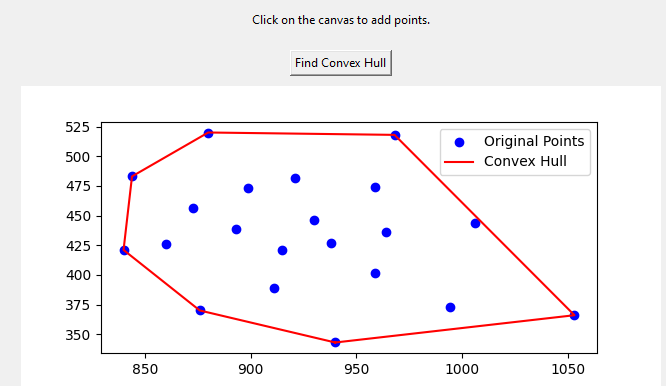
import tkinter as tk  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
import matplotlib.pyplot as plt  
  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x, self.y = float(x), float(y)  
  
 def \_\_eq\_\_(self, other):  
 return self.x == other.x and self.y == other.y  
  
 def \_\_ne\_\_(self, other):  
 return not self == other  
  
 def \_\_repr\_\_(self):  
 return f"({self.x}, {self.y})"  
  
  
def direction(p, q, r):  
 return (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)  
  
  
def distance\_sq(p1, p2):  
 return (p1.x - p2.x) \*\* 2 + (p1.y - p2.y) \*\* 2  
  
  
def jarvis\_march(points):  
 a = min(points, key=lambda point: point.x)  
 index = points.index(a)  
  
 l = index  
 result = [a]  
  
 while True:  
 q = (l + 1) % len(points)  
 for i in range(len(points)):  
 if i == l:  
 continue  
 d = direction(points[l], points[i], points[q])  
 if d > 0 or (d == 0 and distance\_sq(points[i], points[l]) > distance\_sq(points[q], points[l])):  
 q = i  
 l = q  
 if l == index:  
 break  
 result.append(points[q])  
  
 return result  
  
  
class JarvisMarchVisualizationApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Jarvis March Visualization")  
  
 self.points = []  
  
 # Canvas for drawing points and convex hull  
 self.canvas = tk.Canvas(self.master, width=600, height=600, bg="white")  
 self.canvas.pack(expand=tk.YES, fill=tk.BOTH)  
  
 # Label for instructions  
 self.label = tk.Label(self.master, text="Click on the canvas to add points.")  
 self.label.pack(pady=10)  
  
 # Button to find and plot convex hull  
 self.find\_hull\_button = tk.Button(self.master, text="Find Convex Hull", command=self.find\_and\_plot\_convex\_hull)  
 self.find\_hull\_button.pack(pady=10)  
  
 # Matplotlib figure for convex hull plot  
 self.fig, self.ax = plt.subplots()  
 self.canvas\_tkagg = FigureCanvasTkAgg(self.fig, master=self.master)  
 self.canvas\_tkagg\_widget = self.canvas\_tkagg.get\_tk\_widget()  
 self.canvas\_tkagg\_widget.pack()  
  
 # Bind mouse events to canvas  
 self.canvas.bind("<Button-1>", self.add\_point)  
  
 def add\_point(self, event):  
 x, y = event.x, event.y  
 self.points.append(Point(x, y))  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="blue")  
  
 def find\_and\_plot\_convex\_hull(self):  
 if len(self.points) < 3:  
 self.label.config(text="At least 3 points are required.")  
 else:  
 hull\_points = jarvis\_march(self.points)  
 hull\_points.append(hull\_points[0]) # Closing the loop  
  
 # Clear canvas before plotting  
 self.canvas.delete("all")  
  
 # Plot the original points  
 for point in self.points:  
 self.canvas.create\_oval(point.x - 3, point.y - 3, point.x + 3, point.y + 3, fill="blue")  
  
 # Plot the convex hull  
 self.canvas.create\_line(\*[(point.x, point.y) for point in hull\_points], fill='red')  
  
 # Plot convex hull using Matplotlib  
 self.ax.clear()  
 self.ax.scatter(\*zip(\*[(point.x, point.y) for point in self.points]), color='blue', label='Original Points')  
 self.ax.plot(\*zip(\*[(point.x, point.y) for point in hull\_points]), color='red', label='Convex Hull')  
 self.ax.legend()  
 self.canvas\_tkagg.draw()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = JarvisMarchVisualizationApp(root)  
 root.mainloop()

A graph with red lines and blue dots

Description automatically generated

GRAHAM SCAN

import tkinter as tk  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from math import atan2  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x, self.y = float(x), float(y)  
  
 def \_\_eq\_\_(self, other):  
 return self.x == other.x and self.y == other.y  
  
 def \_\_ne\_\_(self, other):  
 return not self == other  
  
 def \_\_repr\_\_(self):  
 return f"({self.x}, {self.y})"  
  
  
def direction(p, q, r):  
 return (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y)  
  
  
def distance\_sq(p1, p2):  
 return (p1.x - p2.x) \*\* 2 + (p1.y - p2.y) \*\* 2  
  
  
def find\_min\_y(points):  
 miny = float('inf')  
 mini = 0  
 for i, point in enumerate(points):  
 if point.y < miny:  
 miny = point.y  
 mini = i  
 if point.y == miny:  
 if point.x < points[mini].x:  
 mini = i  
 return points[mini], mini  
  
  
def graham\_scan(points):  
 p0, index = find\_min\_y(points)  
 points[0], points[index] = points[index], points[0]  
  
 sorted\_polar = sorted(points[1:], key=lambda p: (atan2(p.y - p0.y, p.x - p0.x), p))  
  
 to\_remove = []  
 for i in range(len(sorted\_polar) - 1):  
 d = direction(sorted\_polar[i], sorted\_polar[i + 1], p0)  
 if d == 0:  
 to\_remove.append(i)  
 sorted\_polar = [i for j, i in enumerate(sorted\_polar) if j not in to\_remove]  
  
 m = len(sorted\_polar)  
 if m < 2:  
 print('Convex hull is empty')  
 return []  
  
 stack = []  
 stack.append(points[0])  
 stack.append(sorted\_polar[0])  
 stack.append(sorted\_polar[1])  
  
 for i in range(2, m):  
 while True:  
 d = direction(stack[-2], stack[-1], sorted\_polar[i])  
 if d < 0:  
 break  
 else:  
 stack.pop()  
 stack.append(sorted\_polar[i])  
  
 return stack  
  
  
class GrahamScanVisualizationApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Graham's Scan Visualization")  
  
 self.points = []  
  
 # Canvas for drawing points  
 self.canvas = tk.Canvas(self.master, width=600, height=600, bg="white")  
 self.canvas.pack(expand=tk.YES, fill=tk.BOTH)  
  
 # Label for instructions  
 self.label = tk.Label(self.master, text="Click on the canvas to add points.")  
 self.label.pack(pady=10)  
  
 # Button to find and plot convex hull  
 self.find\_hull\_button = tk.Button(self.master, text="Find Convex Hull", command=self.find\_and\_plot\_convex\_hull)  
 self.find\_hull\_button.pack(pady=10)  
  
 # Matplotlib figure for convex hull plot  
 self.fig, self.ax = plt.subplots()  
 self.canvas\_tkagg = FigureCanvasTkAgg(self.fig, master=self.master)  
 self.canvas\_tkagg\_widget = self.canvas\_tkagg.get\_tk\_widget()  
 self.canvas\_tkagg\_widget.pack()  
  
 # Bind mouse events to canvas  
 self.canvas.bind("<Button-1>", self.add\_point)  
  
 def add\_point(self, event):  
 x, y = event.x, event.y  
 self.points.append(Point(x, y))  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="blue")  
  
 def find\_and\_plot\_convex\_hull(self):  
 if len(self.points) < 3:  
 self.label.config(text="At least 3 points are required.")  
 else:  
 convex\_hull = graham\_scan(self.points)  
  
 # Clear canvas before plotting  
 self.canvas.delete("all")  
  
 # Plot the original points  
 for point in self.points:  
 self.canvas.create\_oval(point.x - 3, point.y - 3, point.x + 3, point.y + 3, fill="blue")  
  
 # Plot the convex hull  
 for i in range(len(convex\_hull) - 1):  
 p1, p2 = convex\_hull[i], convex\_hull[i + 1]  
 self.canvas.create\_line(p1.x, p1.y, p2.x, p2.y, fill='red')  
  
 # Plot convex hull using Matplotlib  
 hull\_x, hull\_y = zip(\*[(point.x, point.y) for point in convex\_hull])  
 self.ax.clear()  
 self.ax.scatter(\*zip(\*[(point.x, point.y) for point in self.points]), color='blue', label='Original Points')  
 self.ax.plot(hull\_x + (hull\_x[0],), hull\_y + (hull\_y[0],), color='red', label='Convex Hull')  
 self.ax.legend()  
 self.canvas\_tkagg.draw()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = GrahamScanVisualizationApp(root)  
 root.mainloop()

****

**Kirkpatrick–Seidel algorithm**

pip install matplotlib mplcursors

import matplotlib.pyplot as plt  
import numpy as np  
import mplcursors  
  
def orientation(p, q, r):  
 val = (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1])  
 if val == 0:  
 return 0 # collinear  
 return 1 if val > 0 else 2 # 1 for clockwise, 2 for counterclockwise  
  
def on\_hull(p, q, r):  
 return orientation(p, q, r) == 2 # Check if the point is on the upper hull  
  
def kirkpatrick\_seidel(points):  
 n = len(points)  
 if n < 3:  
 return points  
  
 points = sorted(points, key=lambda x: x[0])  
  
 upper\_hull = [points[0], points[1]]  
 lower\_hull = [points[0], points[1]]  
  
 for i in range(2, n):  
 upper\_hull.append(points[i])  
 while len(upper\_hull) > 2 and not on\_hull(upper\_hull[-3], upper\_hull[-2], upper\_hull[-1]):  
 upper\_hull.pop(-2)  
  
 lower\_hull.append(points[i])  
 while len(lower\_hull) > 2 and on\_hull(lower\_hull[-3], lower\_hull[-2], lower\_hull[-1]):  
 lower\_hull.pop(-2)  
  
 convex\_hull = upper\_hull + lower\_hull[1:-1]  
  
 return convex\_hull  
  
def onclick(event):  
 x, y = event.xdata, event.ydata  
 points.append((x, y))  
 ax.plot(x, y, marker='o', color='blue')  
 fig.canvas.draw()  
  
 if len(points) > 2:  
 # Compute convex hull using Kirkpatrick and Seidel algorithm  
 convex\_hull = kirkpatrick\_seidel(points)  
  
 # Clear previous hull  
 for line in hull\_lines:  
 line.remove()  
 hull\_lines.clear()  
  
 # Plot the convex hull  
 for i in range(len(convex\_hull)):  
 hull\_lines.append(ax.plot([convex\_hull[i][0], convex\_hull[(i + 1) % len(convex\_hull)][0]],  
 [convex\_hull[i][1], convex\_hull[(i + 1) % len(convex\_hull)][1]],  
 color='red', linewidth=2, linestyle='-')[0])  
  
 fig.canvas.draw()  
  
# Initialize an empty list to store points  
points = []  
  
# Create a figure and axis  
fig, ax = plt.subplots()  
ax.set\_title('Click to add points')  
  
# Connect the click event to the onclick function  
cid = fig.canvas.mpl\_connect('button\_press\_event', onclick)  
  
# List to store hull lines for removal  
hull\_lines = []  
  
mplcursors.cursor(hover=True)  
plt.show()

A screenshot of a graph

Description automatically generated

QUICK ELIMINATION

import tkinter as tk  
import numpy as np  
  
class Point:  
 def \_\_init\_\_(self, x, y):  
 self.x, self.y = x, y  
  
def find\_side(p1, p2, p):  
 return np.sign((p.y - p1.y) \* (p2.x - p1.x) - (p2.y - p1.y) \* (p.x - p1.x))  
  
def line\_dist(p1, p2, p):  
 return np.abs((p.y - p1.y) \* (p2.x - p1.x) - (p2.y - p1.y) \* (p.x - p1.x))  
  
def quick\_hull\_recursive(p1, p2, points):  
 if not points:  
 return []  
  
 # Find the point with the maximum distance from the line formed by p1 and p2  
 max\_dist = -1  
 farthest\_point = None  
  
 for point in points:  
 temp\_dist = line\_dist(p1, p2, point)  
 if temp\_dist > max\_dist:  
 max\_dist = temp\_dist  
 farthest\_point = point  
  
 # Find the points on the left and right side of the line  
 left\_points = [point for point in points if find\_side(p1, farthest\_point, point) > 0]  
 right\_points = [point for point in points if find\_side(farthest\_point, p2, point) > 0]  
  
 # Recursively find the convex hull on both sides  
 hull\_upper = quick\_hull\_recursive(p1, farthest\_point, left\_points)  
 hull\_lower = quick\_hull\_recursive(farthest\_point, p2, right\_points)  
  
 # Combine the results  
 return [p1] + hull\_upper + [farthest\_point] + hull\_lower  
  
def quick\_hull(points):  
 if len(points) < 3:  
 return points  
  
 # Find the leftmost and rightmost points  
 min\_point = min(points, key=lambda p: p.x)  
 max\_point = max(points, key=lambda p: p.x)  
  
 # Divide the points into two sets, upper and lower  
 upper = [point for point in points if find\_side(min\_point, max\_point, point) > 0]  
 lower = [point for point in points if find\_side(max\_point, min\_point, point) > 0]  
  
 # Find the convex hull on both sides  
 hull\_upper = quick\_hull\_recursive(min\_point, max\_point, upper)  
 hull\_lower = quick\_hull\_recursive(max\_point, min\_point, lower)  
  
 # Combine the results  
 convex\_hull = hull\_upper + hull\_lower  
 return convex\_hull  
  
class QuickHullVisualizationApp:  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("Quick Hull Visualization")  
  
 self.points = []  
  
 # Canvas for drawing points and convex hull  
 self.canvas = tk.Canvas(self.master, width=600, height=600, bg="white")  
 self.canvas.pack(expand=tk.YES, fill=tk.BOTH)  
  
 # Label for instructions  
 self.label = tk.Label(self.master, text="Click on the canvas to add points.")  
 self.label.pack(pady=10)  
  
 # Button to find and plot convex hull  
 self.find\_hull\_button = tk.Button(self.master, text="Find Convex Hull", command=self.find\_and\_plot\_convex\_hull)  
 self.find\_hull\_button.pack(pady=10)  
  
 # Bind mouse events to canvas  
 self.canvas.bind("<Button-1>", self.add\_point)  
  
 def add\_point(self, event):  
 x, y = event.x, event.y  
 self.points.append(Point(x, y))  
 self.canvas.create\_oval(x - 3, y - 3, x + 3, y + 3, fill="blue")  
  
 def find\_and\_plot\_convex\_hull(self):  
 if len(self.points) < 3:  
 self.label.config(text="At least 3 points are required.")  
 else:  
 convex\_hull = quick\_hull(self.points)  
 convex\_hull.append(convex\_hull[0]) # Closing the loop  
  
 # Clear canvas before plotting  
 self.canvas.delete("all")  
  
 # Plot the original points  
 for point in self.points:  
 self.canvas.create\_oval(point.x - 3, point.y - 3, point.x + 3, point.y + 3, fill="blue")  
  
 # Plot the convex hull  
 self.canvas.create\_line(\*[(point.x, point.y) for point in convex\_hull], fill='red')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = QuickHullVisualizationApp(root)  
 root.mainloop()

