**Санкт-Петербургский политехнический университет Петра Великого**

Институт прикладной математики и механики

Кафедра «Прикладная математика»

# 

# 

**Отчет по дисциплине «Вычислительные комплексы» по лабораторной работе №1  
«Расстояние Фреше»**

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### 

##### Выполнил студент группы 3630102/60201 Чепулис М.А.

Преподаватель: Баженов А.Н.

Санкт-Петербург

2019

Оглавление

[Постановка задачи 3](#_Toc19552753)

[Теория 3](#_Toc19552754)

[Реализация 3](#_Toc19552755)

[Результаты 4](#_Toc19552756)

[Вычисление Расстояния Фреше для незамкнутых кривых 4](#_Toc19552757)

[Вычисление Расстояния Фреше для «звёздных» множеств 4](#_Toc19552758)

[Обсуждение 5](#_Toc19552759)

[Литература 5](#_Toc19552760)

[Приложение 6](#_Toc19552761)

[Код программы на Python 6](#_Toc19552762)

## Постановка задачи

Построить ломаные кривые

Вычислить для них расстояние Фреше

Показать, на каких элементах реализуется данное расстояние

## Теория

При решении задач, для решения которых необходимо учитывать геометрические свойства множеств, возникает необходимость количественной оценки сходства форм областей.

Рассмотрим метрическое пространство с заданной на нём метрикой-

Стандартный подход к вычислению расстояния Фреше между кривыми- вычисления дискретного расстояния Фреше для ломаных, которые приближают исходные кривые.

Пусть – ломаная кривая

– Ломаная кривая

– Сопряжение между двумя кривыми

Тогда Дискретное расстояние Фреше:

## Реализация

Все задания были выполнены на языке программирования Python в среде разработки PyCharm [[1]](#а16)

Работа с числами и массивами данных осуществлялась при помощи библиотеки Python – NumPy

Графики строились функциями библиотеки Python – matplotlib

## Результаты

### Вычисление Расстояния Фреше для незамкнутых кривых

P = [[0, 0], [4, 2], [6, 5], [12, 6], [15, 7], [15, 10], [18, 13]]

Q = [[1, 1], [2, 5], [7, 7], [8, 12], [13, 14], [15, 16]]

Тогда 7.280109889280518 Между точками P[4] = (15, 7), и Q[4] = (13, 14)

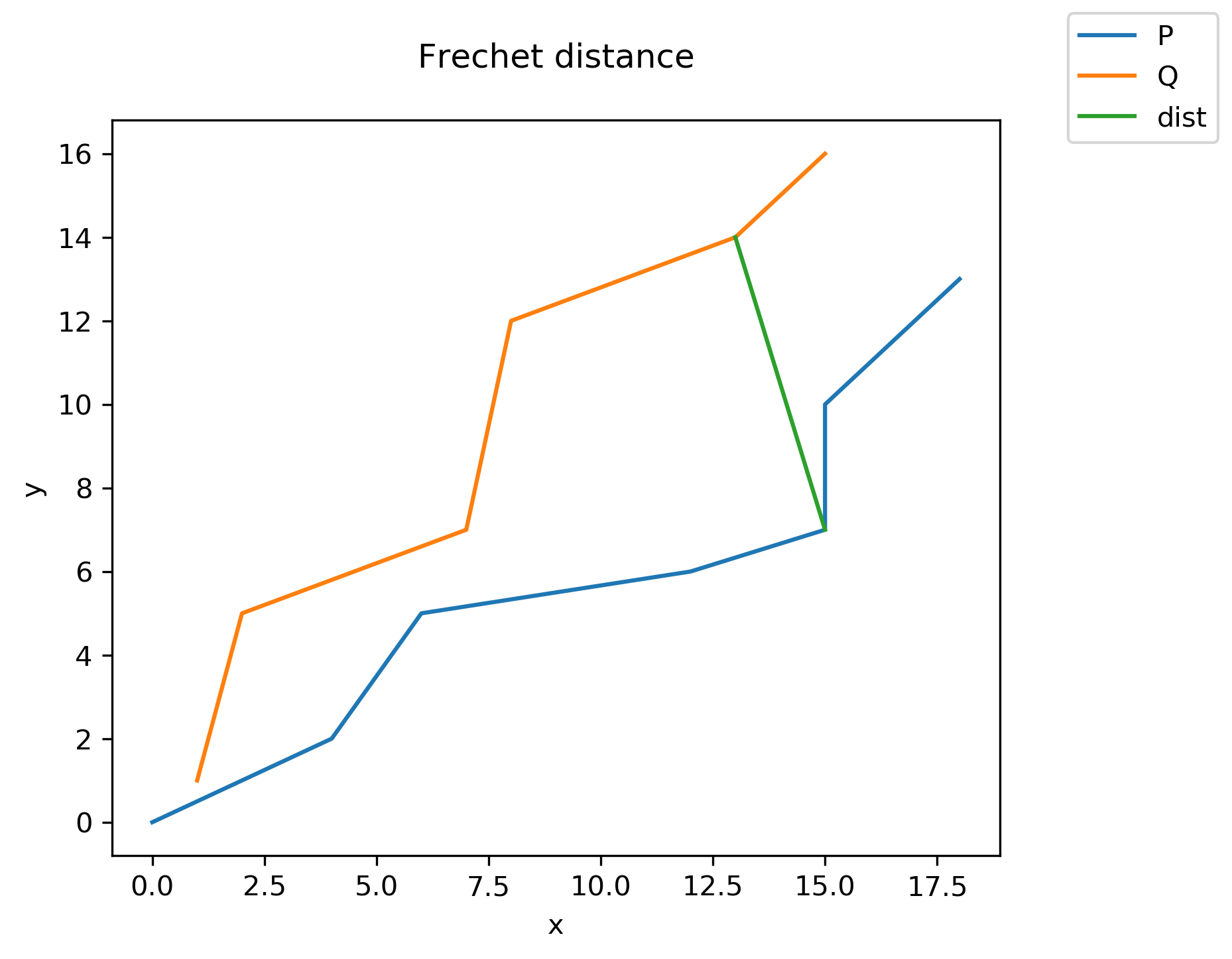


Figure 1 Расстояние Фреше для двух незамкнутых кривых

### Вычисление Расстояния Фреше для «звёздных» множеств

P = [[2, 2], [3, 4], [2, 7], [5, 6], [9, 8], [8, 5], [10, 1], [6, 3], [2, 2]]

Q = [[12, 1], [10, 3], [6, 6], [9, 7], [10, 9], [12, 6], [15, 5], [13, 3], [12, 1]]

Тогда 10.04987562112089 Между точками P[0] = (2, 2), и Q[0] = (12, 1)

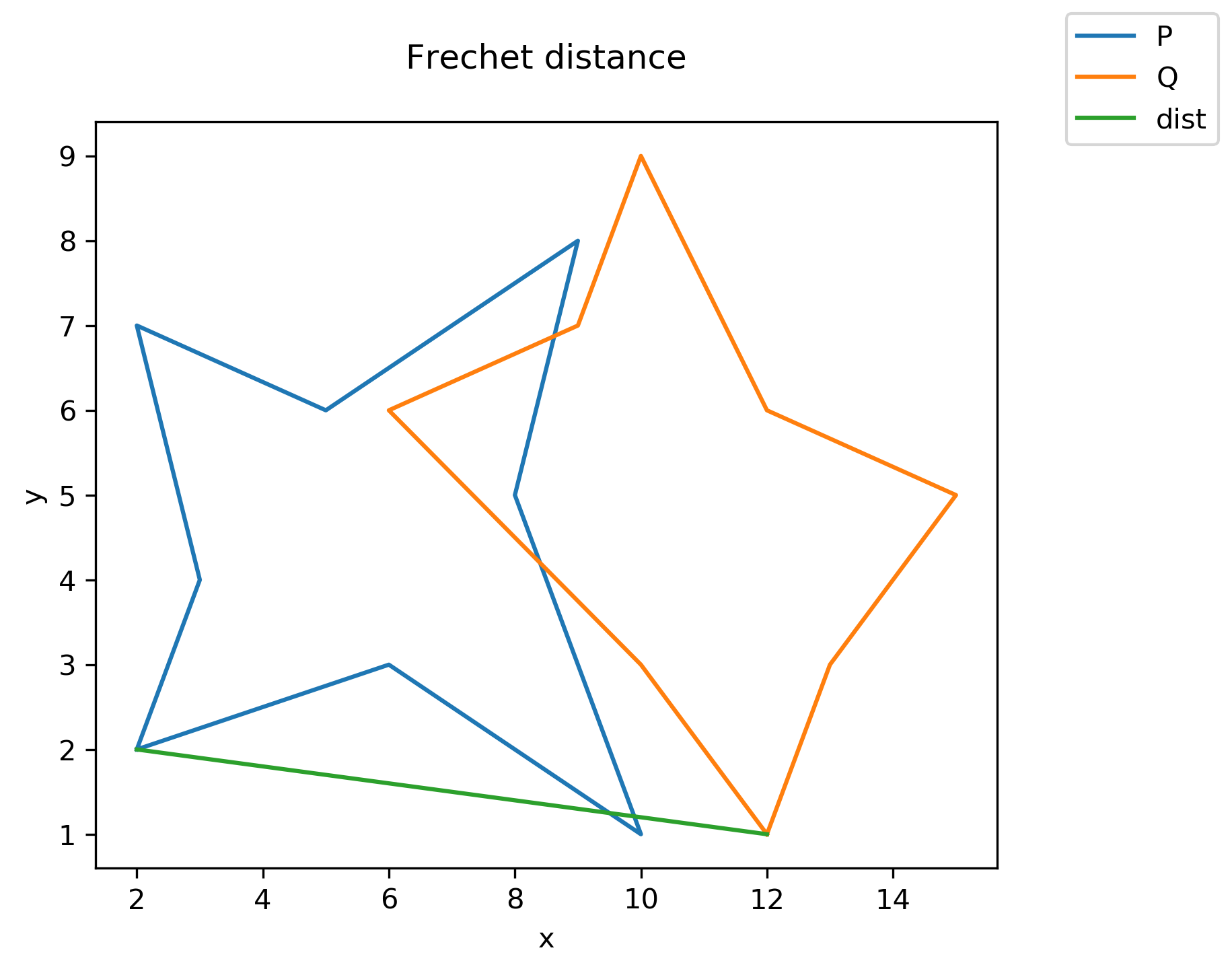


Figure 2 Расстояние Фреше для двух "звёздных" множеств

## Обсуждение

Результаты вычислений совпадают ожидаемыми(примерами)

При данной схеме вычислений наиболее трудоёмкая операция – перебор всех вариантов при вычислении

## Литература

1. Документация библиотеку Python numpy [Электронный ресурс]   
   Режим доступа: <http://www.numpy.org/> (дата обращения сентябрь 2019)
2. Пособие к Лабораторным работам [электронный ресурс, облачное хранилище]

Режим доступа: <https://cloud.mail.ru/public/4ra6/5wwqBzMBC/LabPractics.pdf> (дата обращения сентябрь 2019)

## Приложение

### Код программы на Python

import numpy as np  
import matplotlib.pyplot as plt  
  
  
class DiscrFrechet:  
 def \_\_init\_\_(self, P, Q):  
 self.P = P  
 self.Q = Q  
  
 self.p = len(P)  
 self.q = len(Q)  
  
 self.ca = []  
 self.ind\_matrix = []  
 for i in range(0, self.p):  
 self.ca.append([])  
 self.ind\_matrix.append([])  
 for j in range(0, self.q):  
 self.ca[i].append(-1)  
 self.ind\_matrix[i].append([-1, -1])  
  
  
  
 self.res\_ind = [-1, -1]  
  
 def c(self, i, j):  
 if self.ca[i][j] > -1:  
 return self.ca[i][j]  
 elif i == 0 and j == 0:  
 self.ca[i][j] = self.d(self.P[0], self.Q[0])  
 elif i > 0 and j == 0:  
 self.ca[i][j] = np.max([self.c(i - 1, 0), self.d(self.P[i], self.Q[0])])  
 elif i == 0 and j > 0:  
 self.ca[i][j] = np.max([self.c(0, j - 1), self.d(self.P[0], self.Q[j])])  
 elif i > 0 and j > 0:  
 self.ca[i][j] = np.max([np.min([self.c(i - 1, j), self.c(i, j - 1), self.c(i-1, j-1)]), self.d(self.P[i], self.Q[j])])  
 else:  
 self.ca[i][j] = np.Inf  
 return self.ca[i][j]  
  
 def c\_with\_ind(self, i, j):  
 if self.ca[i][j] > -1:  
 return self.ca[i][j]  
 elif i == 0 and j == 0:  
 self.ca[i][j] = self.d(self.P[0], self.Q[0])  
 self.ind\_matrix[i][j] = [0, 0]  
 elif i > 0 and j == 0:  
 # self.ca[i][j] = np.max([self.c(i - 1, 0), self.d(self.P[i], self.Q[0])])  
 arr = [self.c\_with\_ind(i - 1, 0), self.d(self.P[i], self.Q[0])]  
 arr\_ind = [[i - 1, 0], [i, 0]]  
 ind = np.argmax(arr)  
 self.ca[i][j] = arr[ind]  
 self.ind\_matrix[i][j] = arr\_ind[ind]  
 elif i == 0 and j > 0:  
 # self.ca[i][j] = np.max([self.c(0, j - 1), self.d(self.P[0], self.Q[j])])  
 arr = [self.c\_with\_ind(0, j - 1), self.d(self.P[0], self.Q[j])]  
 arr\_ind = [[0, j - 1], [0, j]]  
 ind = np.argmax(arr)  
 self.ca[i][j] = arr[ind]  
 self.ind\_matrix[i][j] = arr\_ind[ind]  
 elif i > 0 and j > 0:  
 # self.ca[i][j] = np.max([np.min([self.c(i - 1, j), self.c(i, j - 1), self.c(i-1, j-1)]), self.d(self.P[i], self.Q[j])])  
 min\_arr = [self.c\_with\_ind(i - 1, j), self.c\_with\_ind(i, j - 1), self.c\_with\_ind(i - 1, j - 1)]  
 min\_arr\_ind = [[i - 1, j], [i, j - 1], [i - 1, j - 1]]  
 min\_ind = np.argmin(min\_arr)  
 max\_arr = [min\_arr[min\_ind], self.d(self.P[i], self.Q[j])]  
 max\_arr\_ind = [min\_arr\_ind[min\_ind], [i, j]]  
 ind = np.argmax(max\_arr)  
 self.ca[i][j] = max\_arr[ind]  
 self.ind\_matrix[i][j] = max\_arr\_ind[ind]  
 else:  
 self.ca[i][j] = np.Inf  
 return self.ca[i][j]  
  
 def d(self, x, y):  
 sum = 0  
 for i in range(0, len(x)):  
 sum += (x[i] - y[i]) \* (x[i] - y[i])  
 return np.sqrt(sum)  
  
 #def d(self, x, y):  
 # return abs(x - y)  
  
 def get\_ind(self, index):  
 curr\_index = self.ind\_matrix[index[0]][index[1]]  
 if index == curr\_index:  
 return index  
 else:  
 return self.get\_ind(curr\_index)  
  
 def frechet(self):  
 res = self.c\_with\_ind(self.p - 1, self.q - 1)  
 res\_ind = self.get\_ind(self.ind\_matrix[self.p - 1][self.q - 1])  
  
 return res, res\_ind  
 #return self.c(self.p - 1, self.q - 1), self.res\_ind  
  
  
  
def d\_Frechet (P, Q):  
 solver = DiscrFrechet(P, Q)  
  
 return solver.frechet()  
  
  
def process(P, Q, unic\_s):  
 data = [P, Q]  
  
 names = ["P", "Q"]  
  
 dist, d\_index = d\_Frechet(P, Q)  
  
 fig, ax = plt.subplots(nrows=1, ncols=1, sharey=True)  
 ax.set\_xlabel("x")  
 ax.set\_ylabel("y")  
 ax.set\_title("Frechet distance\n")  
 for i in range(0, len(data)):  
 x = []  
 y = []  
 for elem in data[i]:  
 x.append(elem[0])  
 y.append(elem[1])  
 ax.plot(x, y, label=names[i])  
 if d\_index[0] > -1 and d\_index[1] > -1:  
 ax.plot([P[d\_index[0]][0], Q[d\_index[1]][0]], [P[d\_index[0]][1], Q[d\_index[1]][1]], label="dist")  
  
  
 box = ax.get\_position()  
 ax.set\_position([box.x0, box.y0, box.width \* 0.9, box.height])  
  
 fig.legend()  
 fig.savefig("Frechet\_dist%s.png"%unic\_s, dpi=300, format='png', bbox\_inches='tight')  
  
 fig.show()  
 plt.close(fig)  
  
 print(dist, d\_index)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("start")  
 A1 = [[0, 0], [4, 2], [6, 5], [12, 6], [15, 7], [15, 10], [18, 13]]  
 B1 = [[1, 1], [2, 5], [7, 7], [8, 12], [13, 14], [15, 16]]  
 process(A1, B1, "1")  
  
  
 A2 = [[2, 2], [3, 4], [2, 7], [5, 6], [9, 8], [8, 5], [10, 1], [6, 3], [2, 2]]  
 B2 = [[12, 1], [10, 3], [6, 6], [9, 7], [10, 9], [12, 6], [15, 5], [13, 3], [12, 1]]  
 process(A2, B2, "2")  
  
 print("end")