**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*

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- $(V, d)$

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

Пусть $P$: $[0, n] \to V$ – ломаная кривая

$Q$ – Ломаная кривая

$L$– Сопряжение между двумя кривыми

Тогда Дискретное расстояние Фреше:

$$\delta_{dF}(P, Q) = min\|L\|$$

## *Реализация*

Все задания были выполнены на языке программирования Python в среде разработки PyCharm [1]

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 $\delta_{dF}(P, Q) = 7.280109889280518$ Между точками P[4] = (15, 7), и Q[4] = (13, 14)

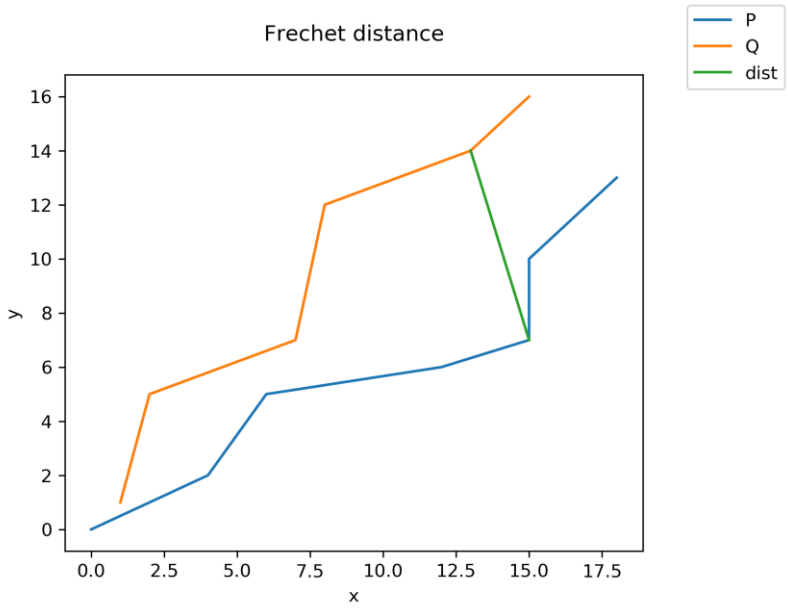


*Figure 1 Расстояние Фреше для двух незамкнутых кривых*

## 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 $\delta_{dF}(P, Q) = 10.04987562112089$ Между точками P[0] = (2, 2), и Q[0] = (12, 1)

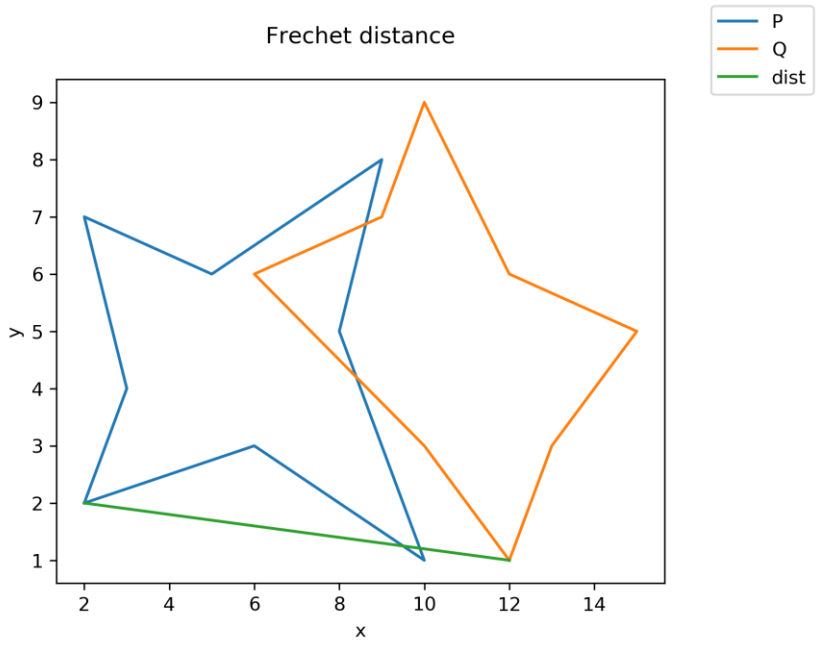


*Figure 2 Расстояние Фреше для двух "звёздных" множеств*

## *Обсуждение*

Результаты вычислений совпадают ожидаемыми(примерами)

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

$$\max\left(min\left(d\left(a_{k_{i-1}}, b_{m_j}\right), d\left(a_{k_{i-1}}, b_{m_{j-1}}\right), d\left(a_{k_i}, b_{m_{j-1}}\right), d\left(a_{k_i}, b_{m_j}\right), \right)\right)$$

## *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")
```