

**Санкт-Петербургский политехнический университет Петра Великого**

Институт прикладной математики и механики

Кафедра «Прикладная математика»

**Отчет по дисциплине «Вычислительные комплексы» по  
лабораторной работе №2  
«Сечение тела вращения»**

Выполнил студент группы 3630102/60201

Чепулис М.А.

Преподаватель: Баженов А.Н.

Санкт-Петербург

2019

## Оглавление

Постановка задачи .....	3
Теория .....	Ошибка! Закладка не определена.
Реализация.....	3
Результаты .....	4
Вычисление Расстояния Фреше для незамкнутых кривых.....	Ошибка! Закладка не определена.
Вычисление Расстояния Фреше для «звёздных» множеств .....	Ошибка! Закладка не определена.
Обсуждение .....	10
Литература .....	11
Приложение.....	Ошибка! Закладка не определена.
Код программы на Python .....	Ошибка! Закладка не определена.

## Постановка задачи

Построить фигуру вращения (тор), как дискретный набор точек в трёхмерном пространстве

[2]

Построить сечение фигуры плоскостью  $x = H$

Т.к. одно из сечений хорошо описывается лемнискатой Бернулли, то нужно построить лемнискату и сравнить её с соответствующим сечением

Варьируя параметры лемнискаты и тора минимизировать расстояние (Фреше) между ними

## Теория

### Сечение тела вращения

Сечение задаётся в плоскости  $XOZ$ . Само тело получается путем вращения всех точек сечения по окружности вокруг от  $Z$

[2]

Так как сечение тела представляет в виде дискретного набора точек, причем каждая из них движется по окружности, то нужно для каждой точки найти пересечение её окружности и плоскостей  $x = H$

Для этого нужно решить систему:

$$\begin{cases} x^2 + y^2 = R^2 \\ x = H \\ z = z' \end{cases}$$

Решением данной системы будет:  $(H, \pm\sqrt{(R^2 - H^2)}, z')$

Если подкоренное выражение меньше нуля, то пересечения нет

### Лемниската Бернулли

Параметрическое уравнение лемнискаты Бернулли:

[2]

$$\begin{cases} y = c \frac{t + t^3}{1 + t^4} \\ x = c \frac{t - t^3}{1 + t^4} \\ z = z' \end{cases}, \text{ где } t = \operatorname{tg}(\varphi), c - \text{переметр}$$

## 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

Трёхмерные графики строились при помощи библиотеки Python - mpl\_toolkits.mplot3d

Модель тела вращения – пары  $(z, r)$ , где  $z$  – координата  $z$ ,  $r$  – радиус вращения точки вокруг оси  $Z$

## Результаты

### Построение сечений

Рассматривается тор со следующими характеристиками:

$R = 20$  – радиус вращения

$r = 10$  – радиус образующей

рассматриваемы плоскости сечения  $x = H$ :

$H = \{0, 5, 10, 15, 20, 30\}$

на графиках оси  $Y$  и  $Z$  могут иметь различный масштаб

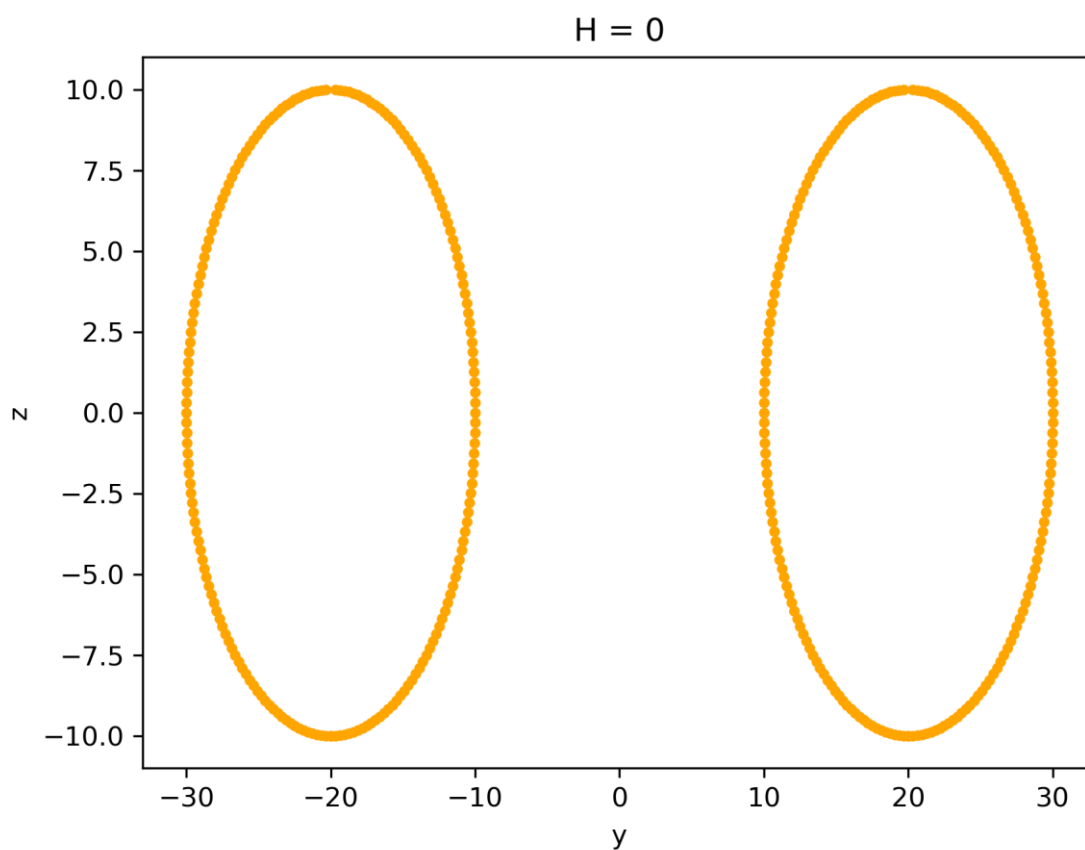


Figure 1 сечение тора плоскость  $x = 0$

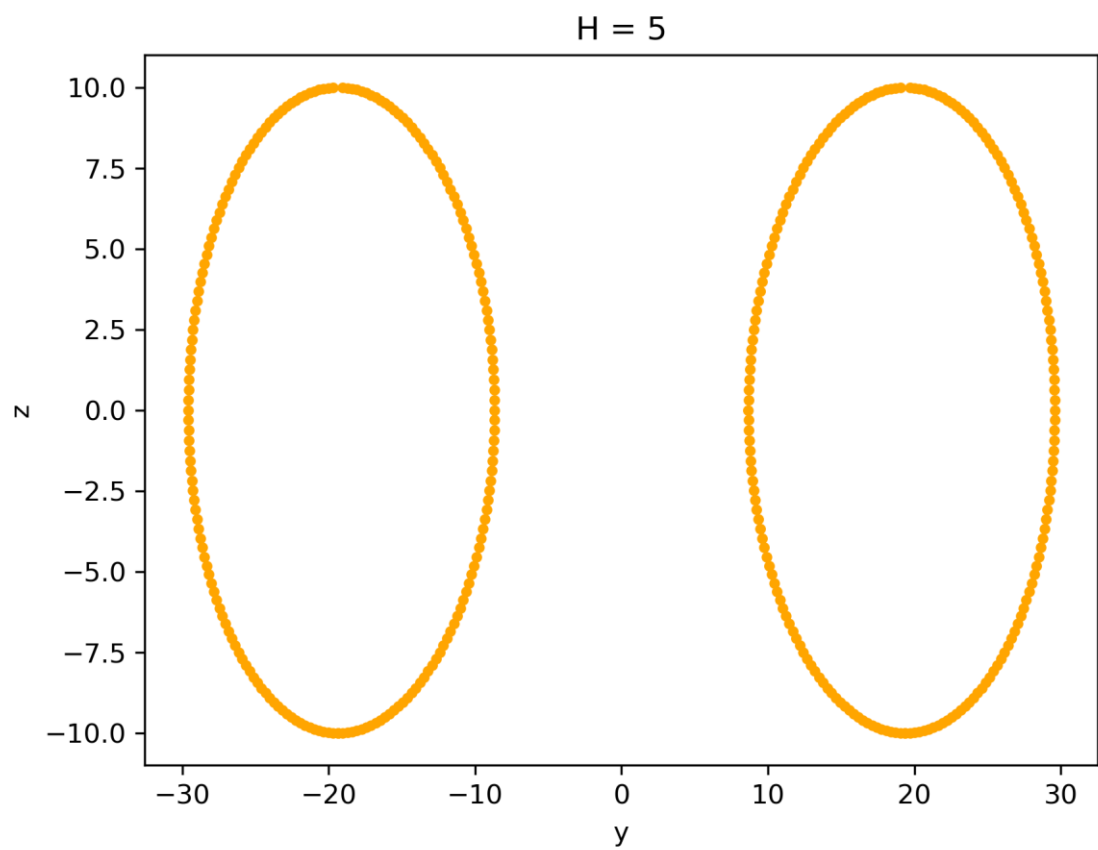


Figure 2 сечение тора плоскость  $x = 5$

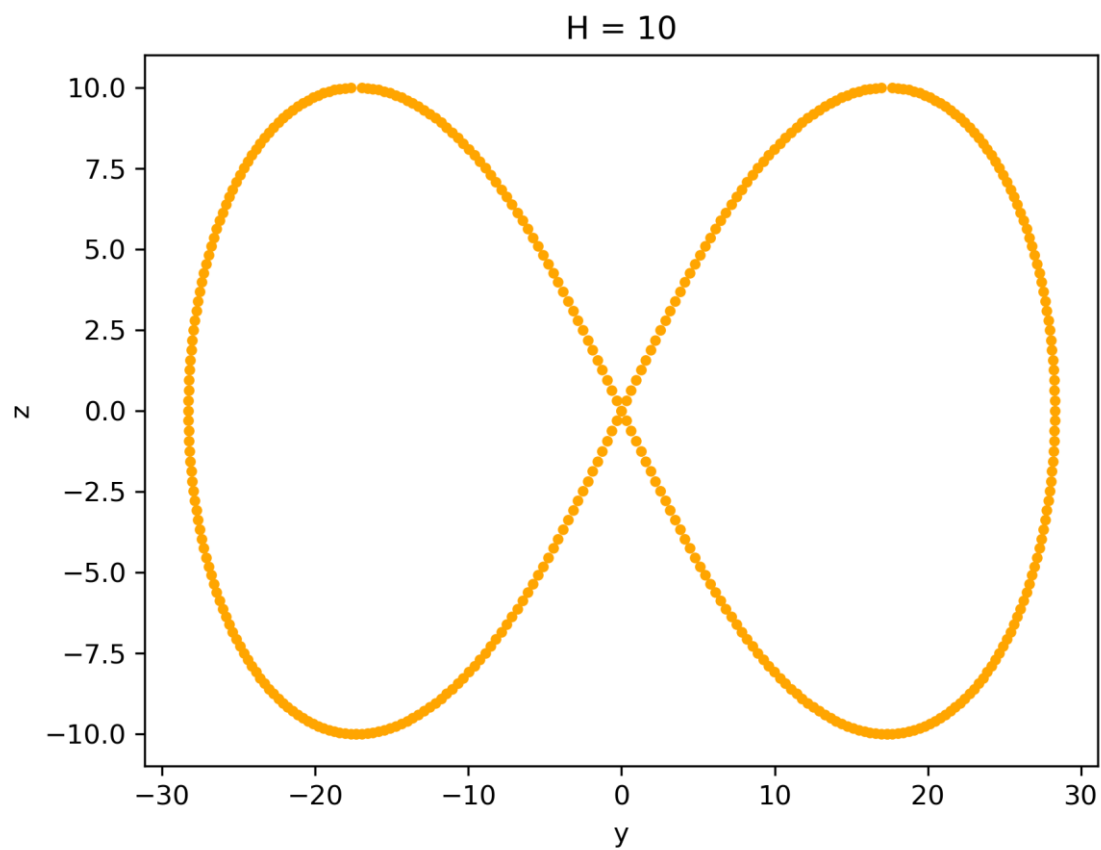


Figure 3 сечение тора плоскость  $x = 10$

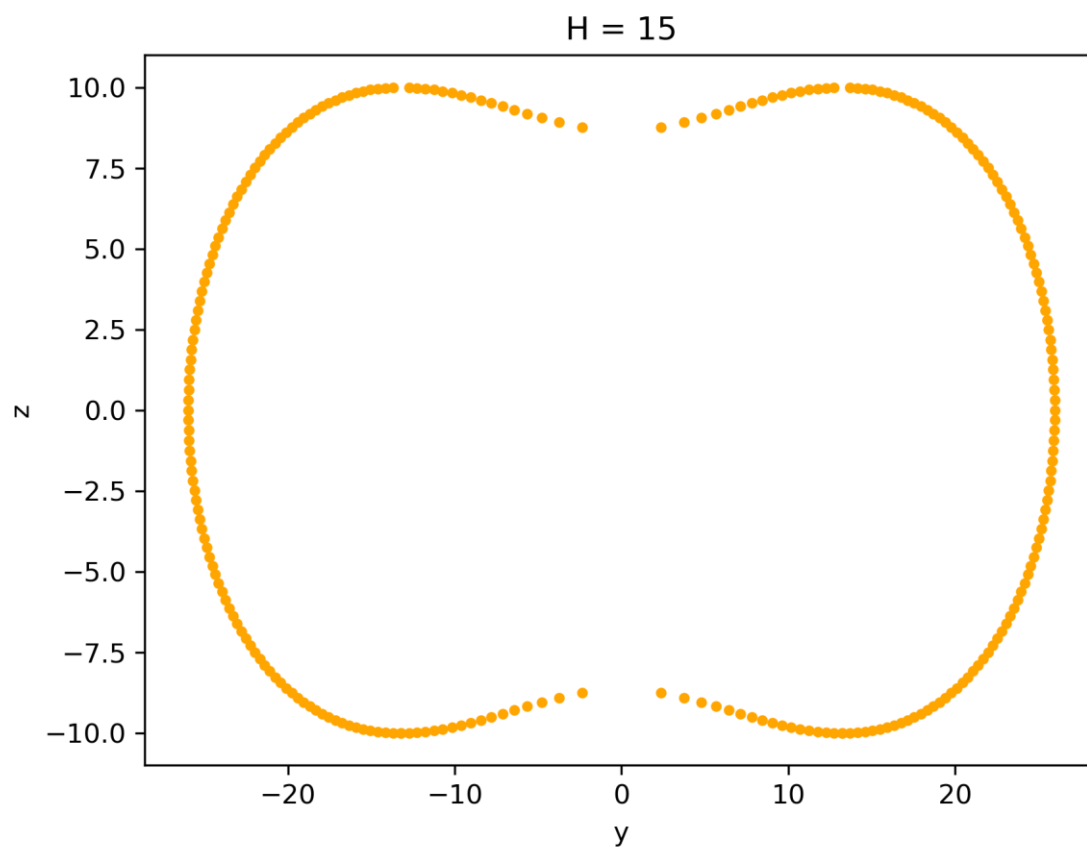


Figure 4 сечение тора плоскость  $x = 15$

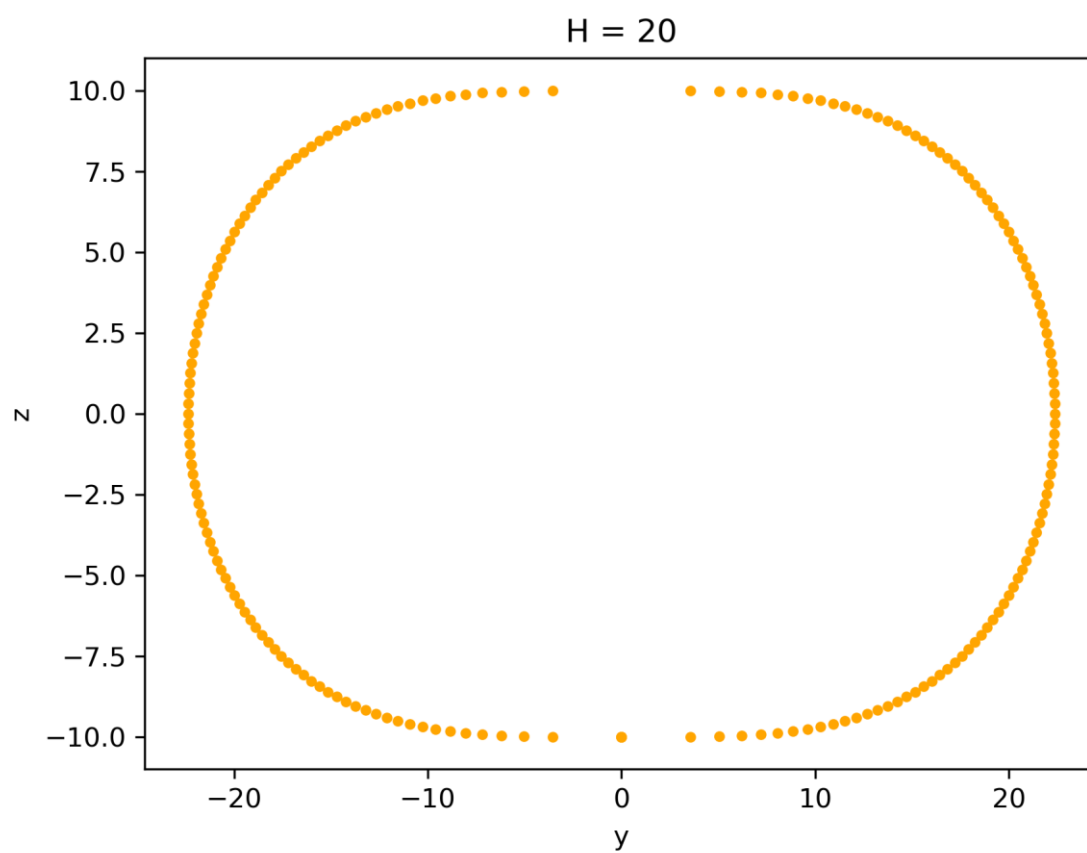


Figure 5 сечение тора плоскость  $x = 20$

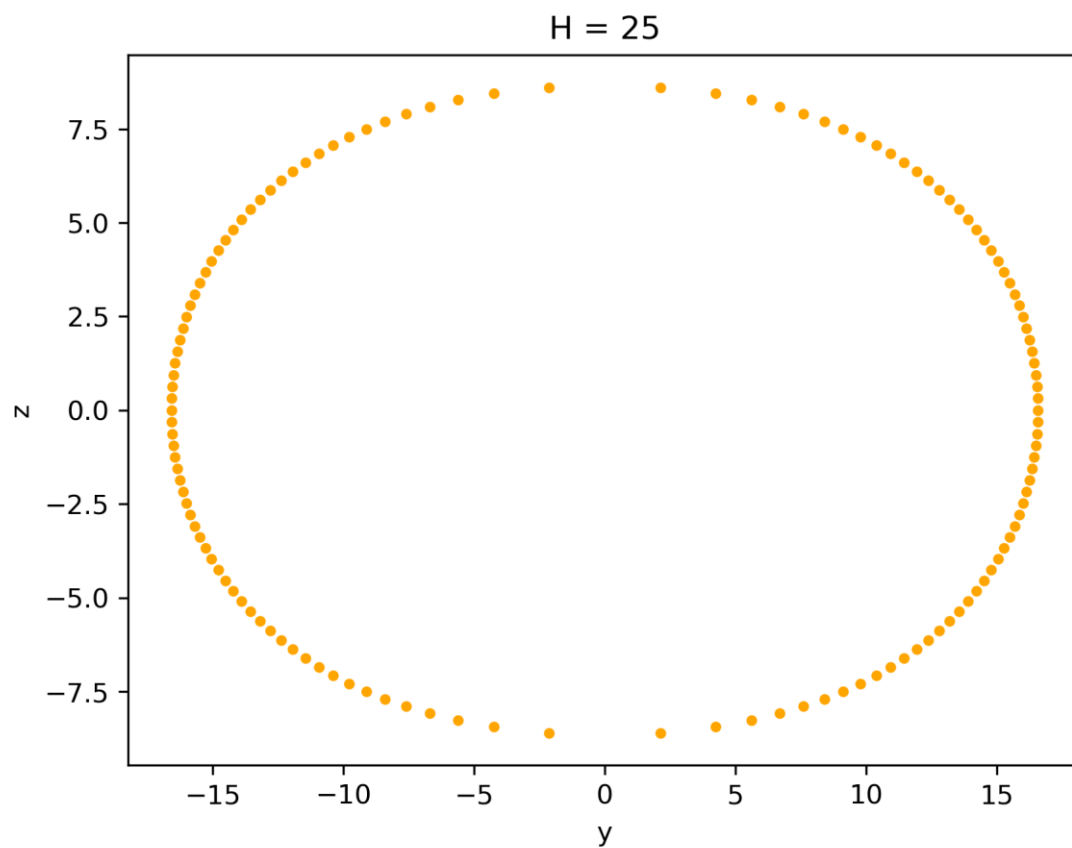


Figure 6 сечение тора плоскость  $x = 25$

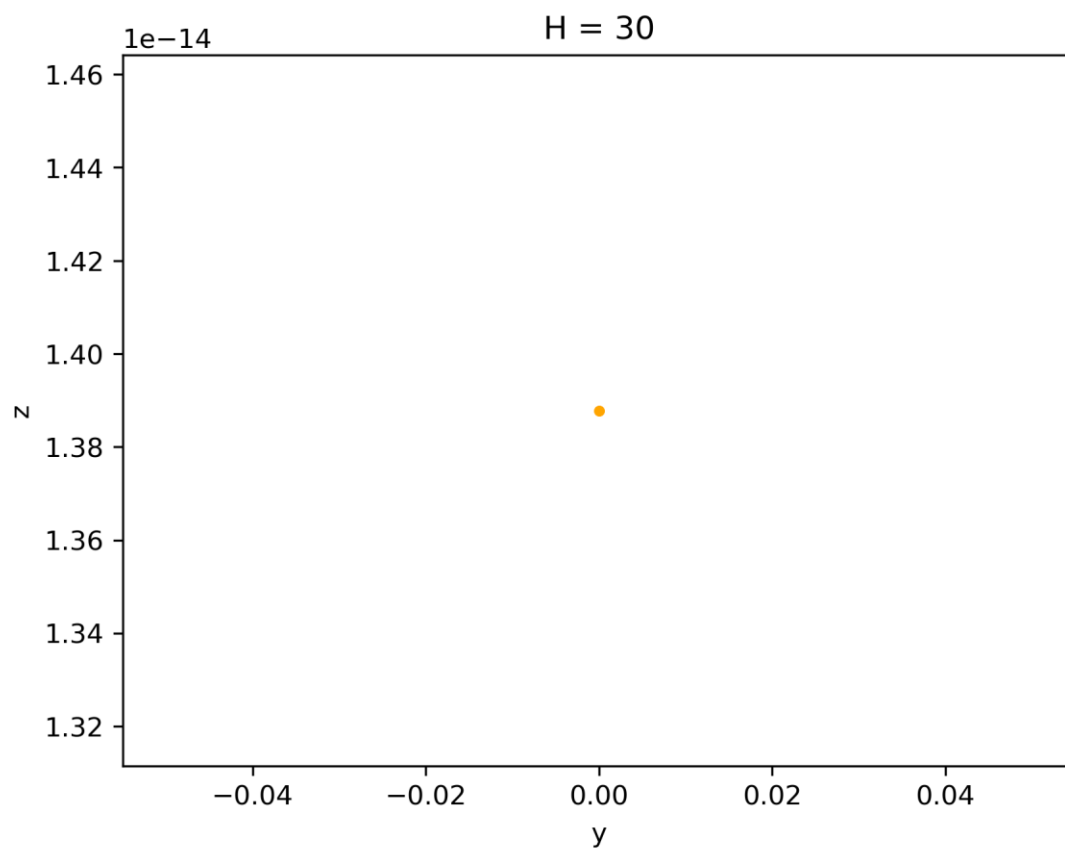


Figure 7 сечение тора плоскость  $x = 30$

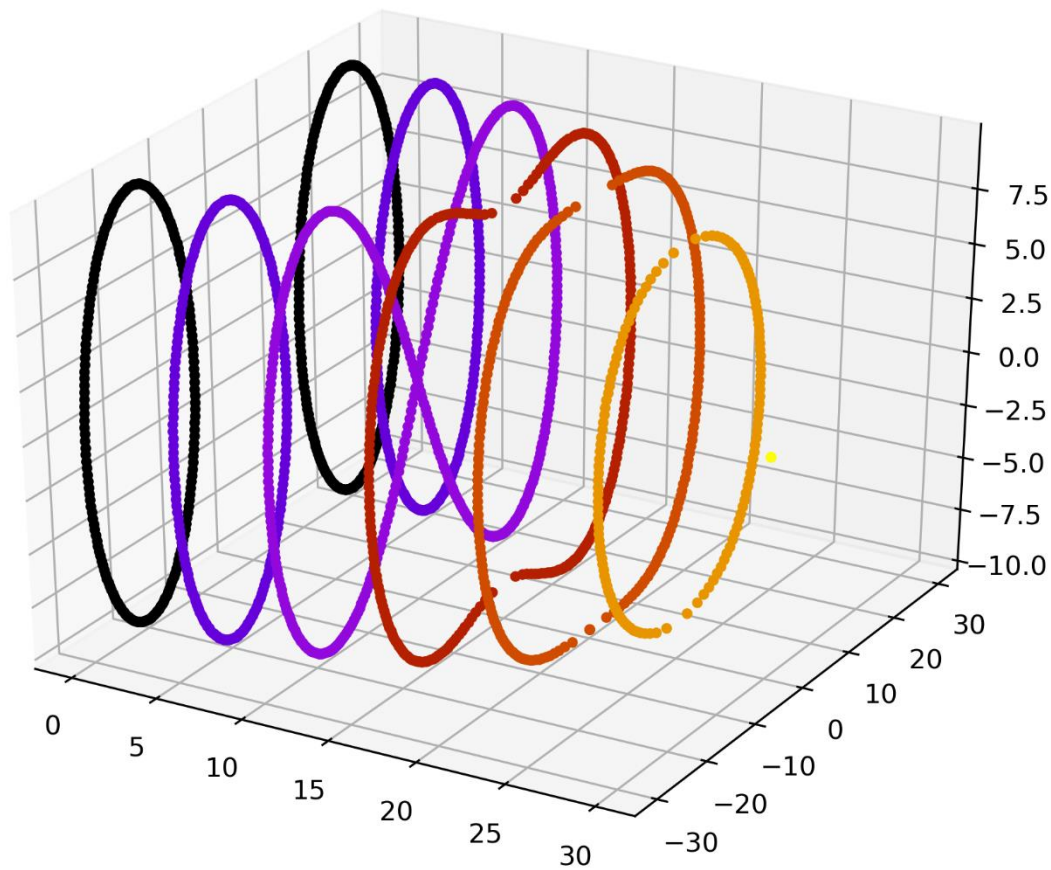


Figure 7 все сечения на одном графике

#### Сравнение сечений с лемниской Бернулли

Рассматривается тор со следующими характеристиками:

$R = 20$  – радиус вращения

$r = 5$  – радиус образующей

Параметр лемнискаты  $c = 20$

Сечение плоскостью  $x = R - r (= 15)$  похоже на лемнискату Бернулли.

Для их сравнения воспользуемся расстоянием Фреше

Расстояние Фреше = 5.3732



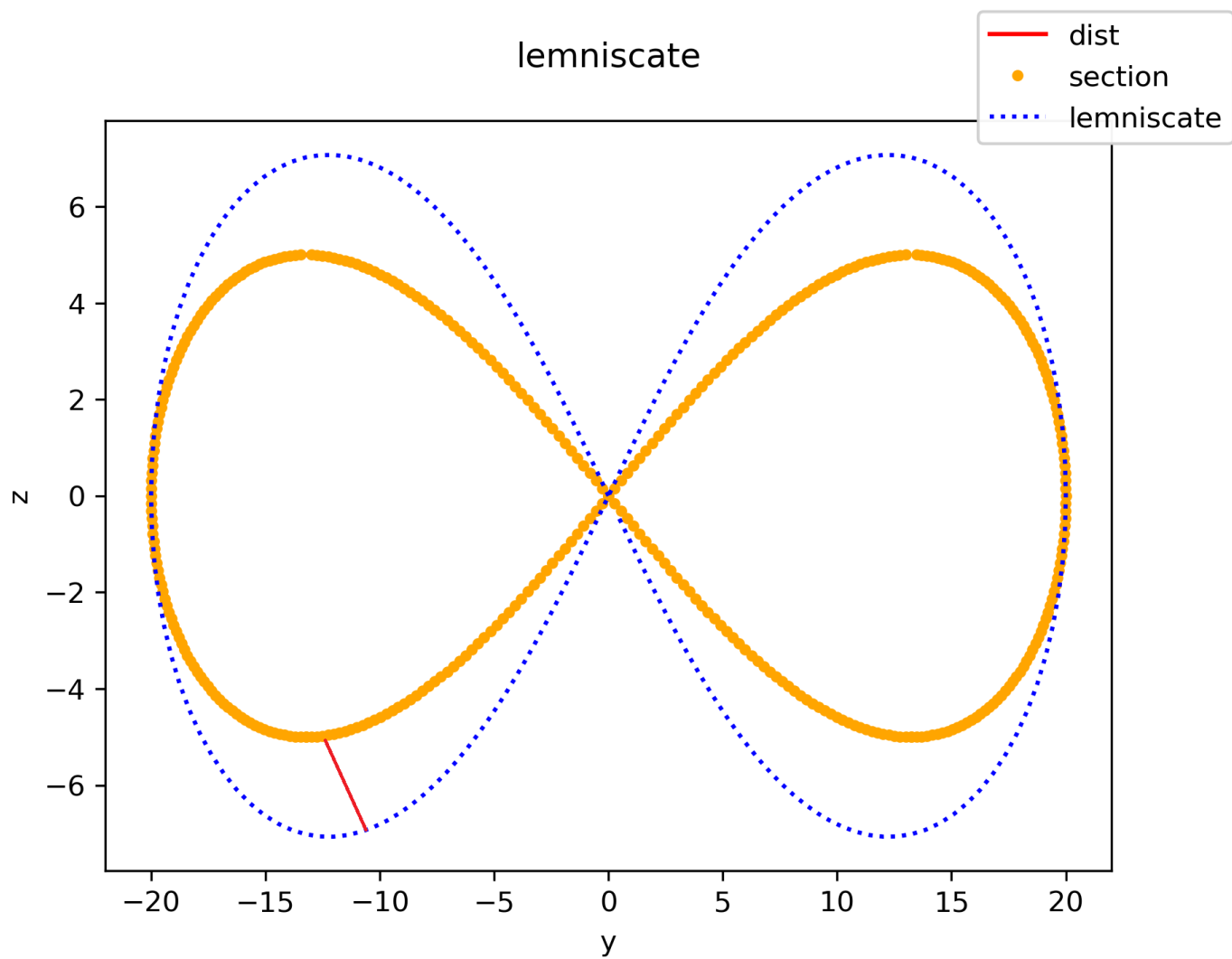


Figure 8 расстояние Фреше между лемнискатой и сечением тора  $x = 15$

Путем подбора параметров тора и лемнискаты добьёмся их практического совпадения:

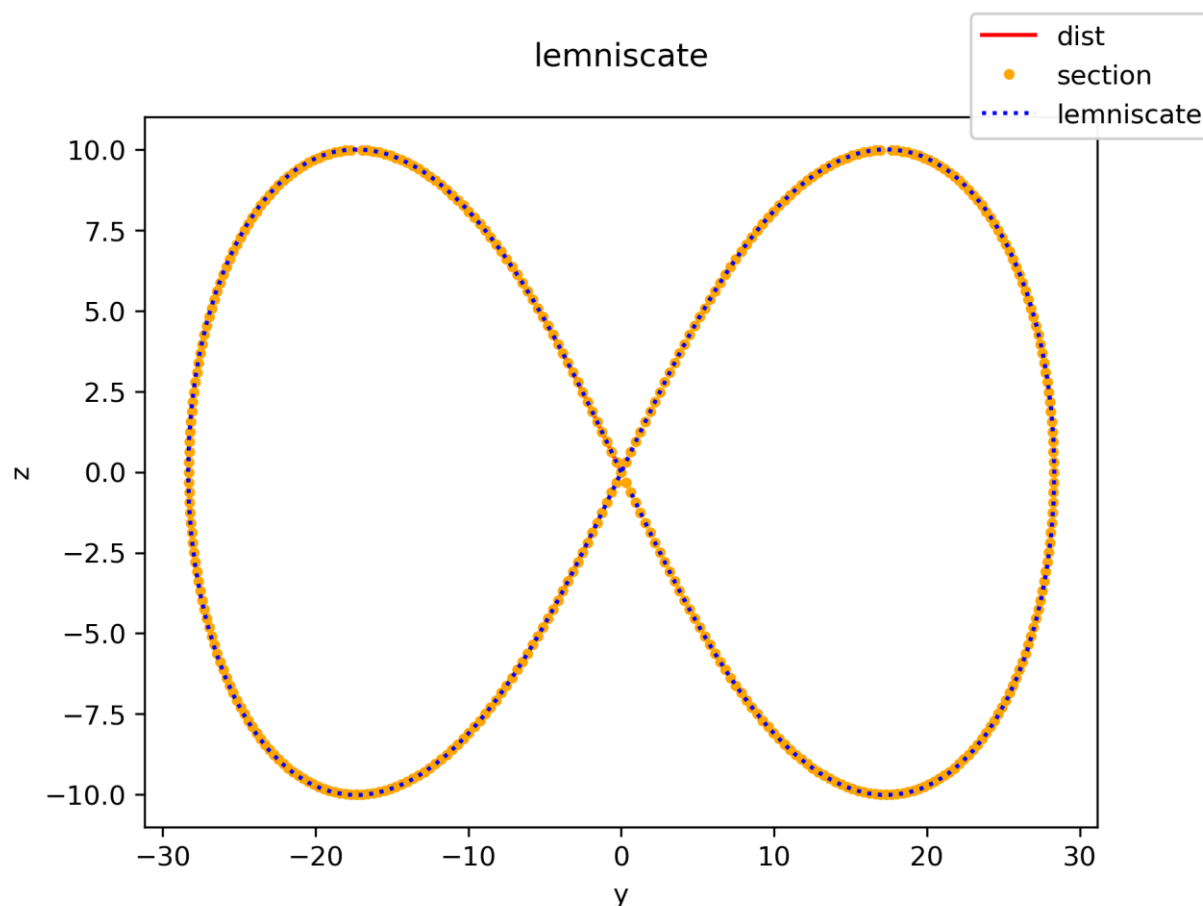
$R = 20$  – радиус вращения

$r = 10$  – радиус образующей

сечение плоскость  $x = 10$

Параметр лемнискаты  $c = 28$

Расстояние Фреше = 0.2381



## Обсуждение

По результатам видно, что в общем случае сечение плоскостью  $x=R-r$  действительно похоже на лемнискату Бернулли, но не совпадает с ней.

Можно подобрать параметры тора и лемнискаты так, чтобы сечение практически совпадало с лемнискатой (расстояние Фреше мало, но не равняется нулю из-за дискретизации фигур)

Для совпадения радиус образующей тора ( $r$ ) должен быть в 2 раза меньше его радиуса вращения ( $R$ )

Параметр лемнискаты – параметр масштаба (не влияет на её форму), следовательно, от него не зависит необходимое соотношение между  $r$  и  $R$

## **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

## Lab\_1.py

```
import pylab
import frechet
import matplotlib.pyplot as plt
# noinspection PyInterpreter
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

class TorPoint:
    def __init__(self, z_, r1_, r2_):
        self.z = z_
        self.r1 = r1_
        self.r2 = r2_

class Point:
    def __init__(self, x_, y_, z_):
        self.x = x_
        self.y = y_
        self.z = z_

class Tor:
    def __init__(self, z_, r_rotate, r_circle):
        self.z = z_
        self.R = r_rotate
        self.r = r_circle
        self.points = []

    def generate_points(self, num_points):
        self.points = []
        n = num_points
        step = np.pi / n
        fi = -np.pi / 2.0

        for i in range(0, n):
            z = self.r * np.sin(fi)
            dr = self.r * np.cos(fi)
            point = TorPoint(z, self.R - dr, self.R + dr)
            self.points.append(point)
            fi += step

    def get_tor_points(self):
        return self.points

def points_on_plane(tor_point, x_plane):
    z = tor_point.z
    r1 = tor_point.r1
    r2 = tor_point.r2

    point1 = Point(x_plane, np.sqrt(r1**2 - x_plane**2), z)
    point2 = Point(x_plane, np.sqrt(r2**2 - x_plane**2), z)

    point3 = Point(x_plane, -np.sqrt(r1**2 - x_plane**2), z)
    point4 = Point(x_plane, -np.sqrt(r2**2 - x_plane**2), z)
    return point1, point2, point3, point4

def get_y_array(points):
```

```

data = []
for point in points:
    data.append(point.y)
return data

def get_z_array(points):
    data = []
    for point in points:
        data.append(point.z)
    return data

def get_intersection_tor_plane(tor, x_plane):
    tor_points = tor.get_tor_points()

    result = []
    left1 = []
    left2 = []
    left3 = []
    left4 = []

    left_jump_2_3_flag = False
    left_jump_1_4_flag = False
    right_jump_2_3_flag = False
    right_jump_1_4_flag = False

    right1 = []
    right2 = []
    right3 = []
    right4 = []
    for point in tor_points:
        tmp = points_on_plane(point, x_plane)

        if not np.isnan(tmp[0].y):
            if not right_jump_1_4_flag:
                right1.append(tmp[0])
            else:
                right4.append(tmp[0])
        else:
            right_jump_1_4_flag = True

        if not np.isnan(tmp[1].y):
            if not right_jump_2_3_flag:
                right2.append(tmp[1])
            else:
                right3.append(tmp[1])
        else:
            right_jump_2_3_flag = True

        if not np.isnan(tmp[2].y):
            if not left_jump_1_4_flag:
                left1.append(tmp[2])
            else:
                left4.append(tmp[2])
        else:
            left_jump_1_4_flag = True

        if not np.isnan(tmp[3].y):
            if not left_jump_2_3_flag:
                left2.append(tmp[3])
            else:
                left3.append(tmp[3])
        else:
            left_jump_2_3_flag = True

```

```

right1.reverse()
right4.reverse()
left1.reverse()
left4.reverse()

left = left2 + left3 + left4 + left1
right = right2 + right3 + right4 + right1
right.reverse()

return left, right

def print_points(points):
    print()
    for point in points:
        print(point.x, point.y, point.z)

def plot_3D(plane_cut, name, filename):

    data = []

    fig = pylab.figure()
    axes = Axes3D(fig)

    i = 0
    number = len(plane_cut)
    cmap = plt.get_cmap('gnuplot')
    colors = [cmap(i) for i in np.linspace(0, 1, number)]
    for points_arr in plane_cut:
        for points in points_arr:
            x = []
            y = []
            z = []
            x1 = []
            y1 = []
            z1 = []
            for elem in points:
                x.append([elem.x])
                y.append([elem.y])
                z.append([elem.z])
                x1.append(elem.x)
                y1.append(elem.y)
                z1.append(elem.z)

            x = np.matrix(x)
            y = np.matrix(y)
            z = np.matrix(z)
            axes.plot(x1, y1, z1, ".", color=colors[i])
            #axes.plot_wireframe(x, y, z, color=colors[i], linewidth=5, linestyle='-')

        i += 1

    fig.savefig(filename, dpi=300, format='png', bbox_inches='tight')
    fig.show()
    plt.close(fig)

def draw_cut(points_arr, name, filename):
    plt.axis('scaled')
    fig, ax = plt.subplots(nrows=1, ncols=1, sharey=True)
    ax.set_xlabel("y")
    ax.set_ylabel("z")
    ax.set_title(name)

```

```

for points in points_arr:
    x = []
    y = []
    z = []
    for elem in points:
        x.append(elem.x)
        y.append(elem.y)
        z.append(elem.z)
    ax.plot(y, z, ".", color="orange")
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width, box.height])
#fig.legend()
fig.savefig(filename, dpi=300, format='png', bbox_inches='tight')

fig.show()
plt.close(fig)

def plane_research(tor, planes):
    data = []
    for plane in planes:
        left, right = get_intersection_tor_plane(tor, plane)
        draw_cut([left, right], "H = %i" % plane, "Tor_section(H = %i).png" % plane)
        data.append([left, right])

    plot_3D(data, "H = %i" % len(planes), "Tor_cuts(H = %i).png" % len(planes))

def lemniscate(c):
    result = []
    n = 360
    fi = 0
    step = 2 * np.pi / n
    left = []
    right = []
    for i in range(0, n//4):
        t = np.tan(fi)
        x = c * (t + t ** 3) / (1 + t ** 4)
        y = c * (t - t ** 3) / (1 + t ** 4)
        right.append([x, y])
        fi += step

    for i in range(n//4, n // 2):
        t = np.tan(fi)
        x = c * (t + t ** 3) / (1 + t ** 4)
        y = c * (t - t ** 3) / (1 + t ** 4)
        left.append([x, y])
        fi += step

    '''
    names = ["left", "right"]
    fig, ax = plt.subplots(nrows=1, ncols=1, sharey=True)
    data = [left, right]
    i = 0
    for points in data:
        x = []
        y = []
        for elem in points:
            x.append(elem[0])
            y.append(elem[1])
        ax.plot(x, y, label=names[i])
        i+=1
    fig.legend()

```

```

fig.show()
'''
return left, right

def draw_line_on_plot(line, ax, color):
    data = [line]
    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="lemniscate", color=color)

def process_lemniscate(tor, plane, c):
    name = "lemniscate"
    filename = "lemniscate.png"

    left, right = get_intersection_tor_plane(tor, plane)
    points_arr = [left, right]
    l, r = lemniscate(c)

    plt.axis('scaled')
    fig, ax = plt.subplots(nrows=1, ncols=1, sharey=True)
    ax.set_xlabel("y")
    ax.set_ylabel("z")
    ax.set_title(name + "\n")

    data = []
    data_arr = []
    draw_flag = True
    for points in points_arr:
        x = []
        y = []
        z = []
        tmp = []
        for elem in points:
            x.append(elem.x)
            y.append(elem.y)
            z.append(elem.z)
            data.append([elem.y, elem.z])
            tmp.append([elem.y, elem.z])
        data_arr.append(tmp)
        if draw_flag:
            ax.plot(y, z, "-", color="red", label="dist")
            ax.plot(y, z, ".", color="orange", label="section")
            draw_flag = False
        else:
            ax.plot(y, z, ".", color="orange")

    draw_line_on_plot(l + r, ax, "b")

    temp_lemn = [r, l]
    temp_data = [data_arr[0], data_arr[1]]
    dist1, ind1 = frechet.d_Frechet(temp_lemn[0], temp_data[1])
    dist2, ind2 = frechet.d_Frechet(temp_lemn[0], temp_data[1])
    '''
    if dist1 > dist2 + 100:
        if ind1[0] > -1 and ind1[1] > -1:
            ax.plot([temp_lemn[0][ind1[0]][0], temp_data[0][ind1[1]][0]],
                    [temp_lemn[0][ind1[0]][1], temp_data[0][ind1[1]][1]], label="dist")

```



```

        print("frechet distance = ", dist1)
    else:
        if ind2[0] > -1 and ind2[1] > -1:
            ax.plot([temp_lemn[1][ind2[0]][0], temp_data[1][ind2[1]][0]],
                    [temp_lemn[1][ind2[0]][1], temp_data[1][ind2[1]][1]], label="dist")
            print("frechet distance = ", dist1)
        '''
    # box = ax.get_position()
    # ax.set_position([box.x0, box.y0, box.width * 0.9, box.height])
    leg = fig.legend()
    fig.legend()
    fig.savefig(filename, dpi=300, format='png', bbox_inches='tight')

    fig.show()
    plt.close(fig)

def main():
    R = 20
    r1 = 10
    numpoints = 360

    tor = Tor(0, R, r1)
    tor.generate_points(100)
    #H = np.array([i for i in range(0, R + r1)])
    #plane_research(tor, H)
    H = [0, 5, 10, 15, 20, 25, 30]
    plane_research(tor, H)
    process_lemniscate(tor, R - r1, R + r1 - r1 / 6)

    print("plane = ", R - r1, "c = ", 28)

if __name__ == "__main__":
    main()

```

## frechet.py

```

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()

count = 0

def draw(P, Q, points, circle=False):
    data = [P, Q]
    names = ["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])
        if(circle):
            x.append(data[elem[0]])
            y.append(data[elem[1]])
        ax.plot(x, y, label=names[i])

    ax.plot([P[points[0], 0], Q[points[1], 0]], [P[points[0], 1], Q[points[1], 1]],
label="dist")

    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.9, box.height])

    fig.legend()
    global count

    fig.savefig("Frechet_dist%s.png" % count, dpi=300, format='png',
bbox_inches='tight')
    count = count + 1

    fig.show()
    plt.close(fig)

def process(P, Q, unic_s):
    data = [P, Q]

    names = ["P", "Q"]

    dist, d_index = d_Frechet(P, Q)

    tmp_P = P.copy()
    tmp_Q = Q.copy()

```

```

while tmp_P.__contains__(P[d_index[0]]):
    tmp_P.remove(P[d_index[0]])

while tmp_Q.__contains__(Q[d_index[1]]):
    tmp_Q.remove(Q[d_index[1]])
tmp_dist, tmp_d_index = d_Frechet(tmp_P, tmp_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")
        if tmp_d_index[0] > -1 and tmp_d_index[1] > -1:
            ax.plot([tmp_P[tmp_d_index[0]][0], tmp_Q[tmp_d_index[1]][0]],
[tmp_P[tmp_d_index[0]][1], tmp_Q[tmp_d_index[1]][1]], label="dist_2")

    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)
print(tmp_dist, tmp_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")
"""

```