**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](#_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 фигуру вращения (тор), как дискретный набор точек в трёхмерном пространстве [[2]](#ref2)

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 X0Z. Само тело получается путем вращения всех точек сечения по окружности вокруг от Z [[2]](#ref2)

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 плоскостиx = H

Для этого нужно решить систему:

Решением данной системы будет:

Если подкоренное выражение меньше нуля, то пересечения нет

### Лемниската Бернулли

Параметрическое уравнение лемнискаты Бернулли: [[2]](#ref2)

## 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

Трехмерные 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

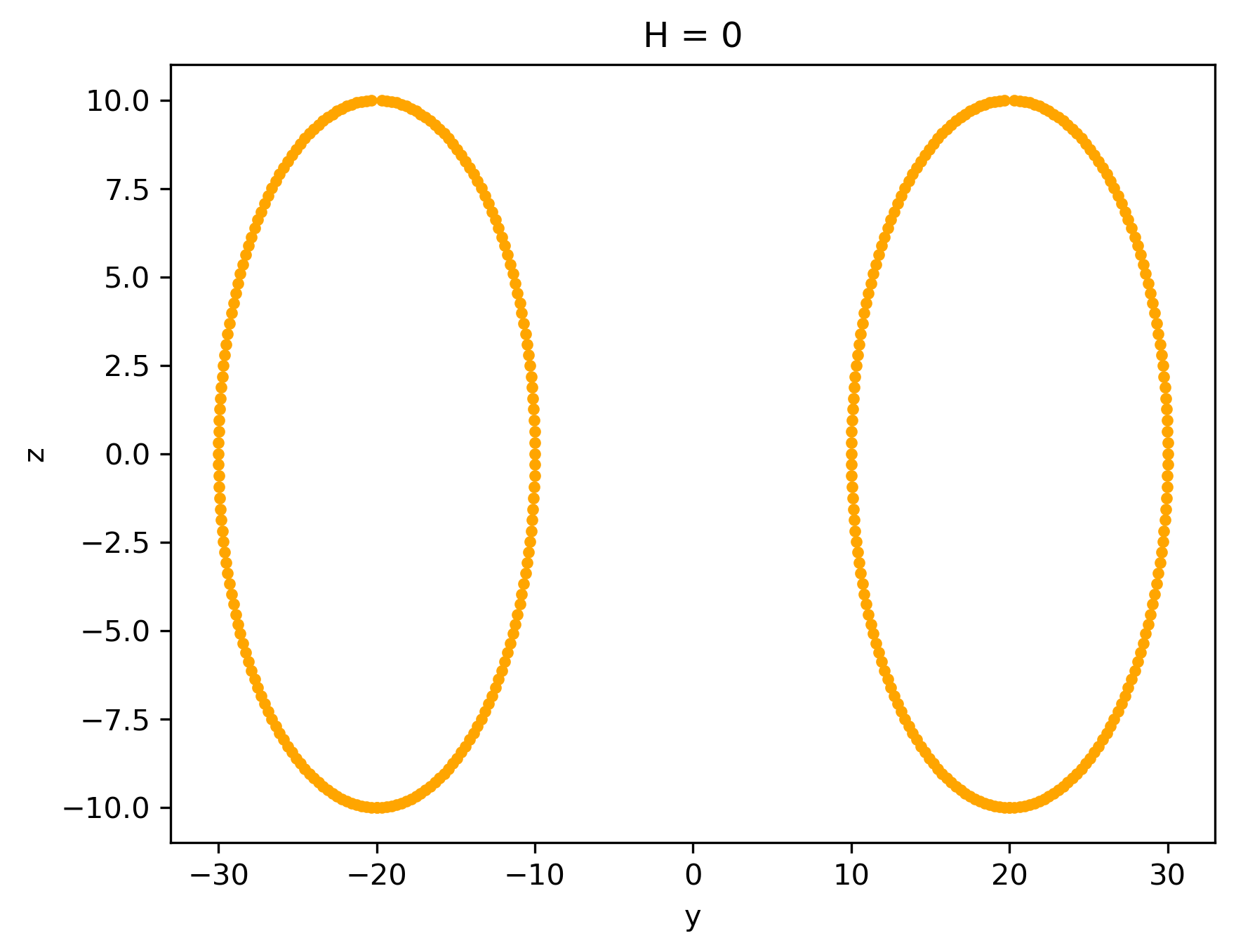


Figure 1 сечение тора плоскость х = 0

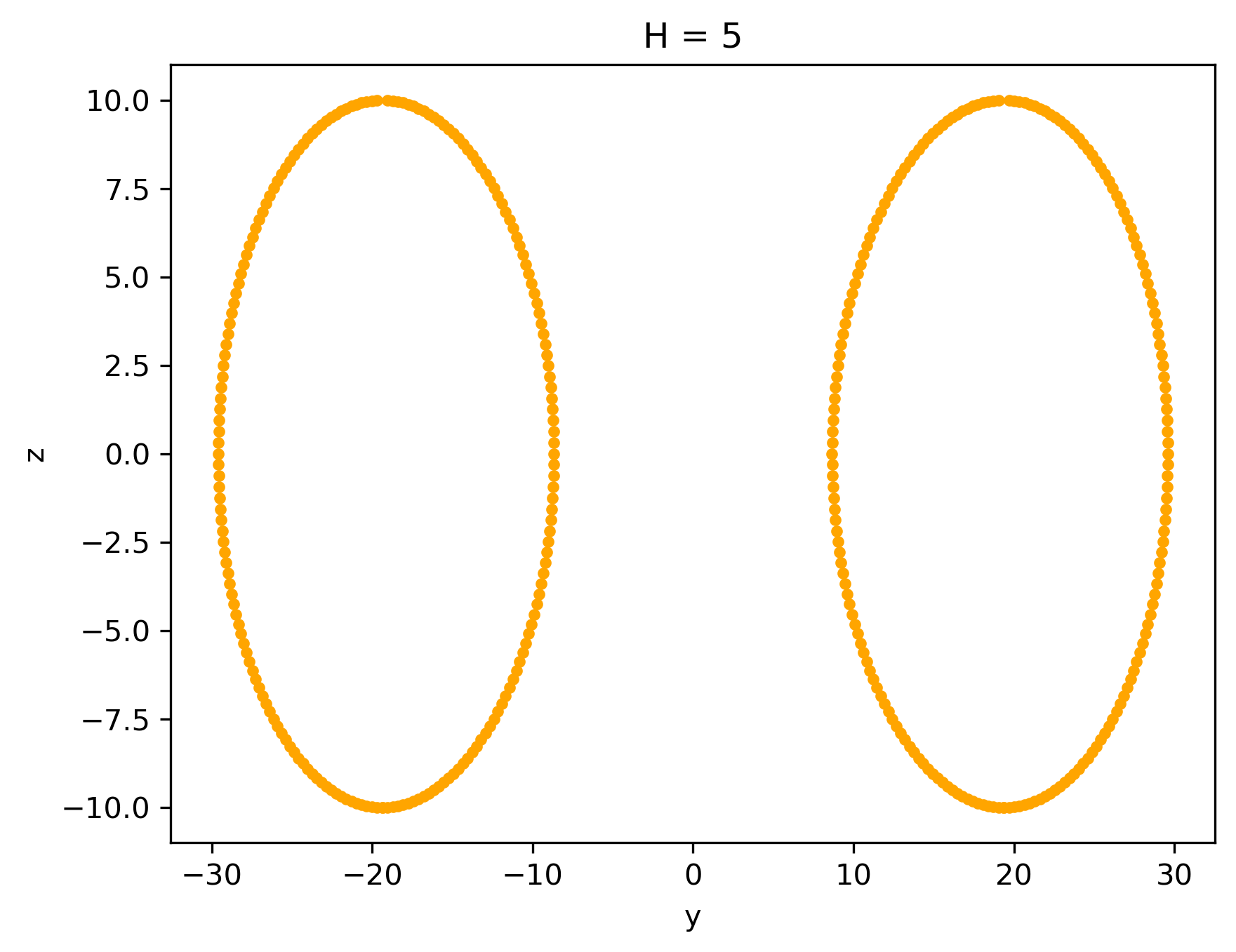


Figure 2 сечение тора плоскость х = 5

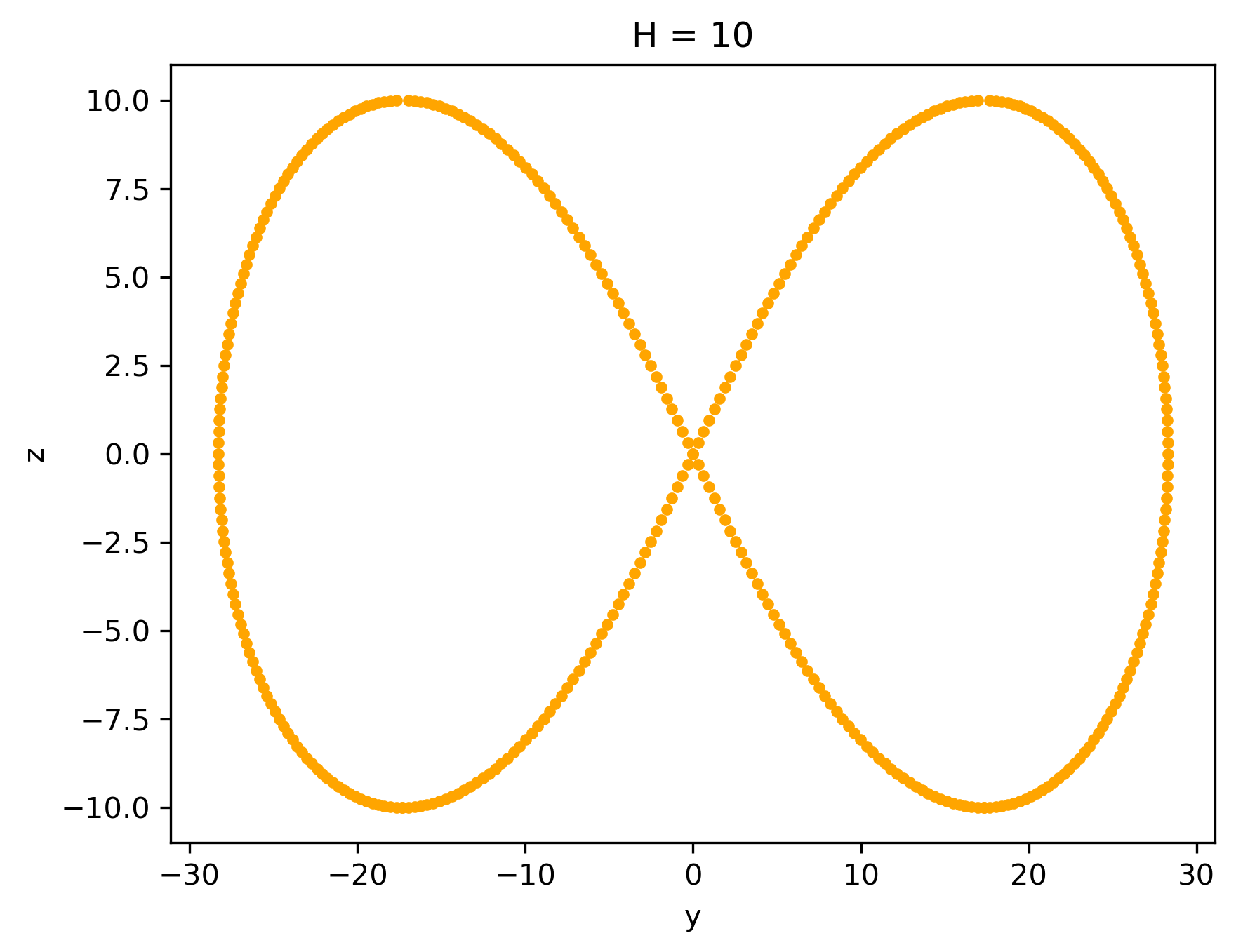


Figure 3 сечение тора плоскость х = 10

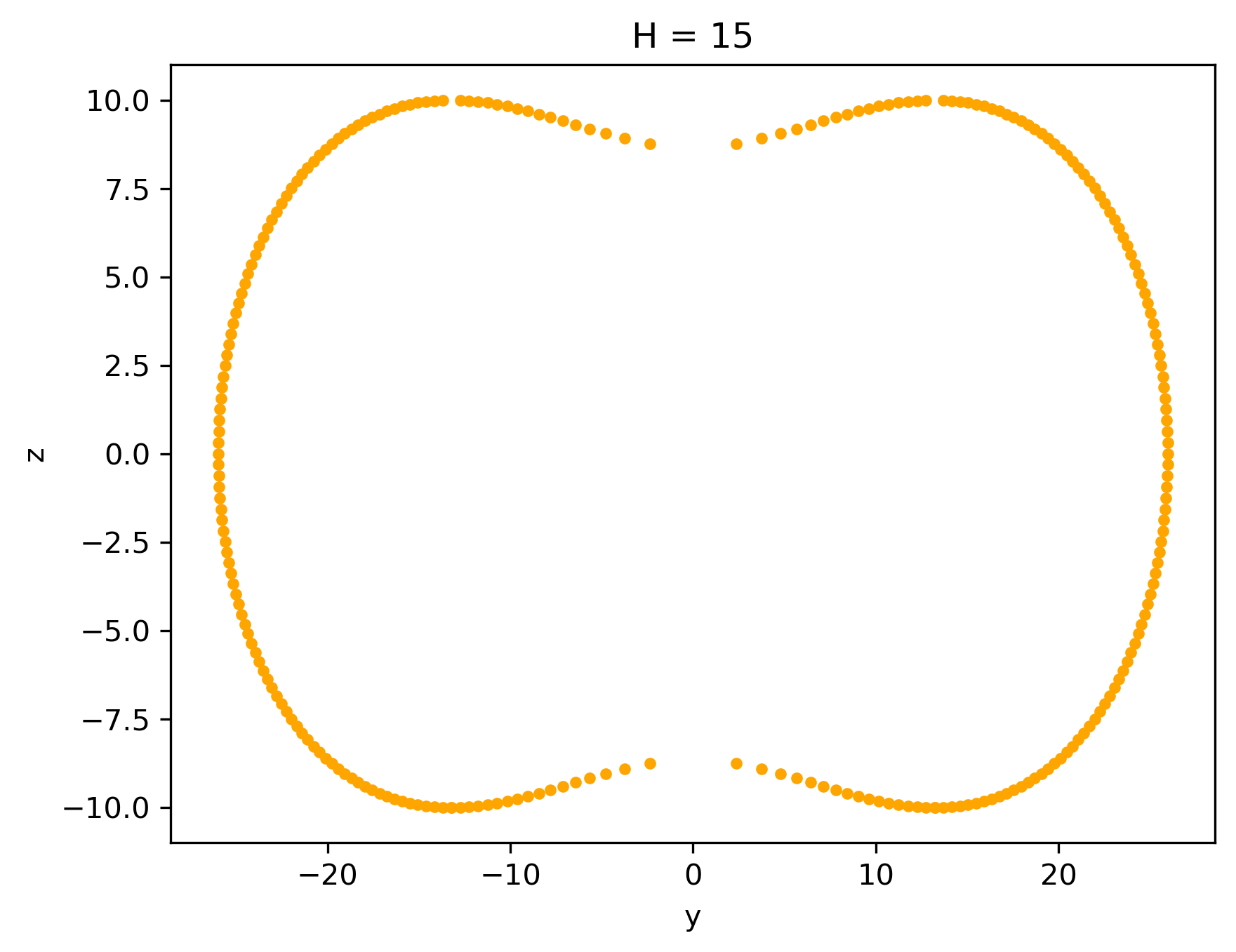


Figure 4 сечение тора плоскость х = 15

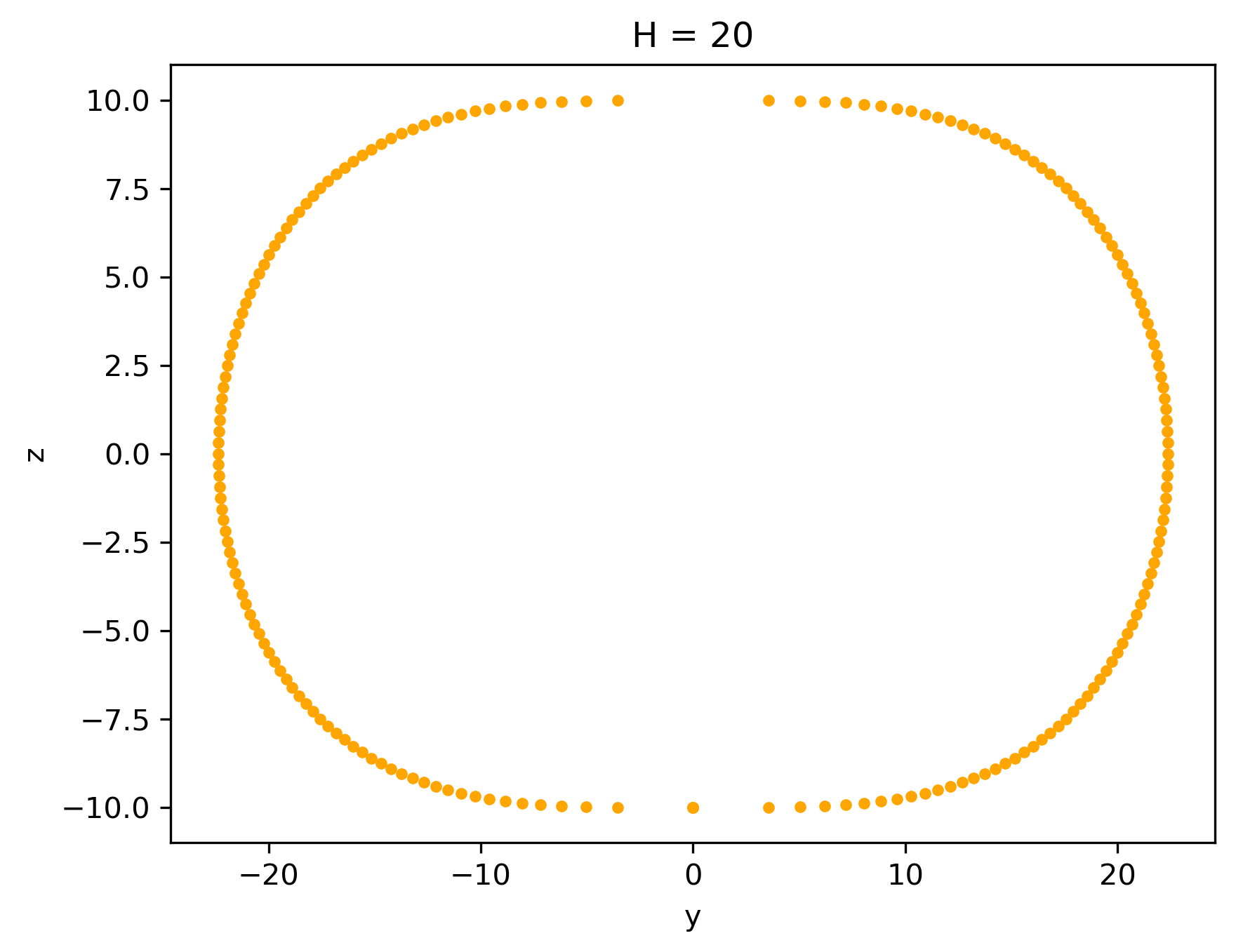


Figure 5 сечение тора плоскость х = 20

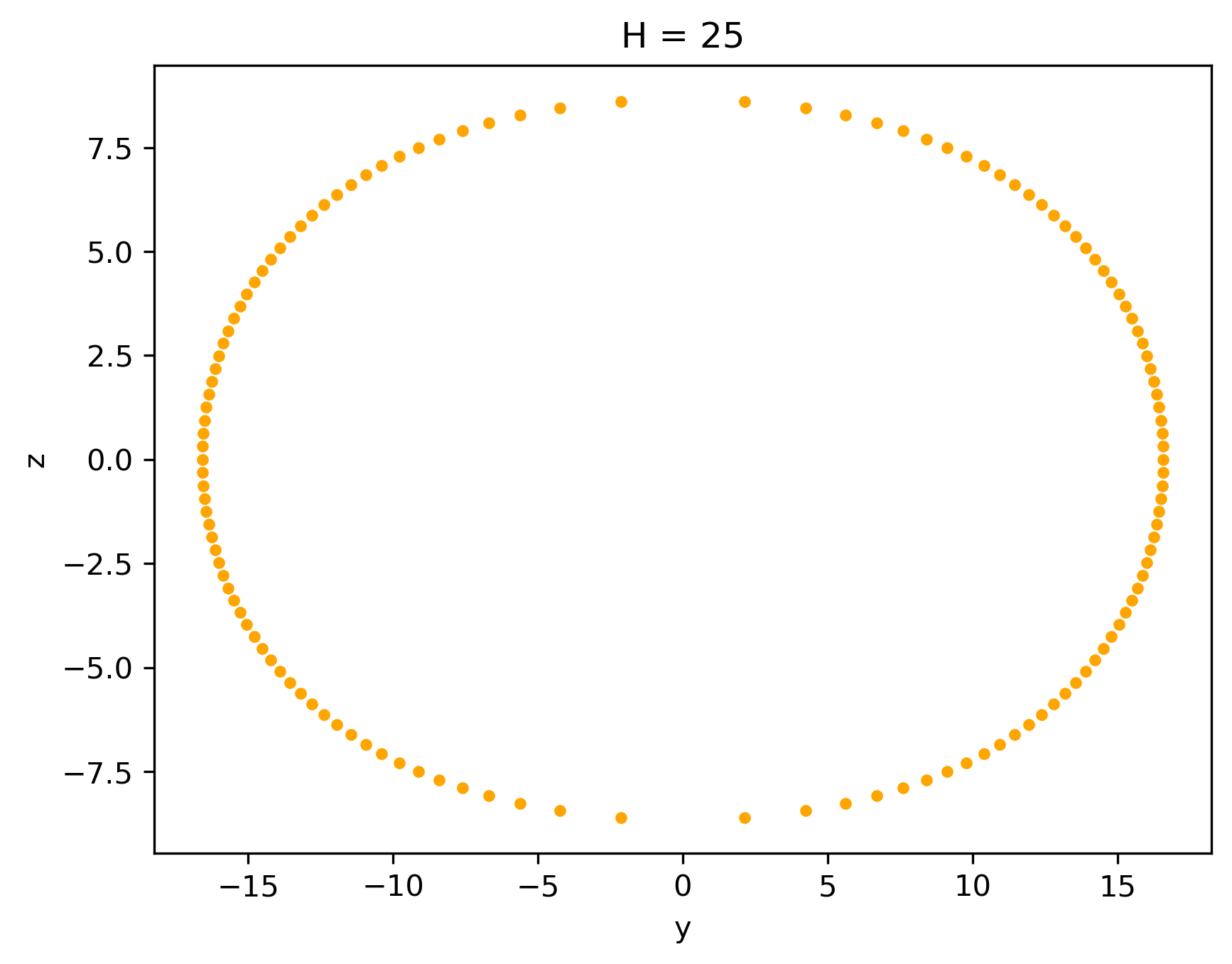


Figure 6 сечение тора плоскость х = 25

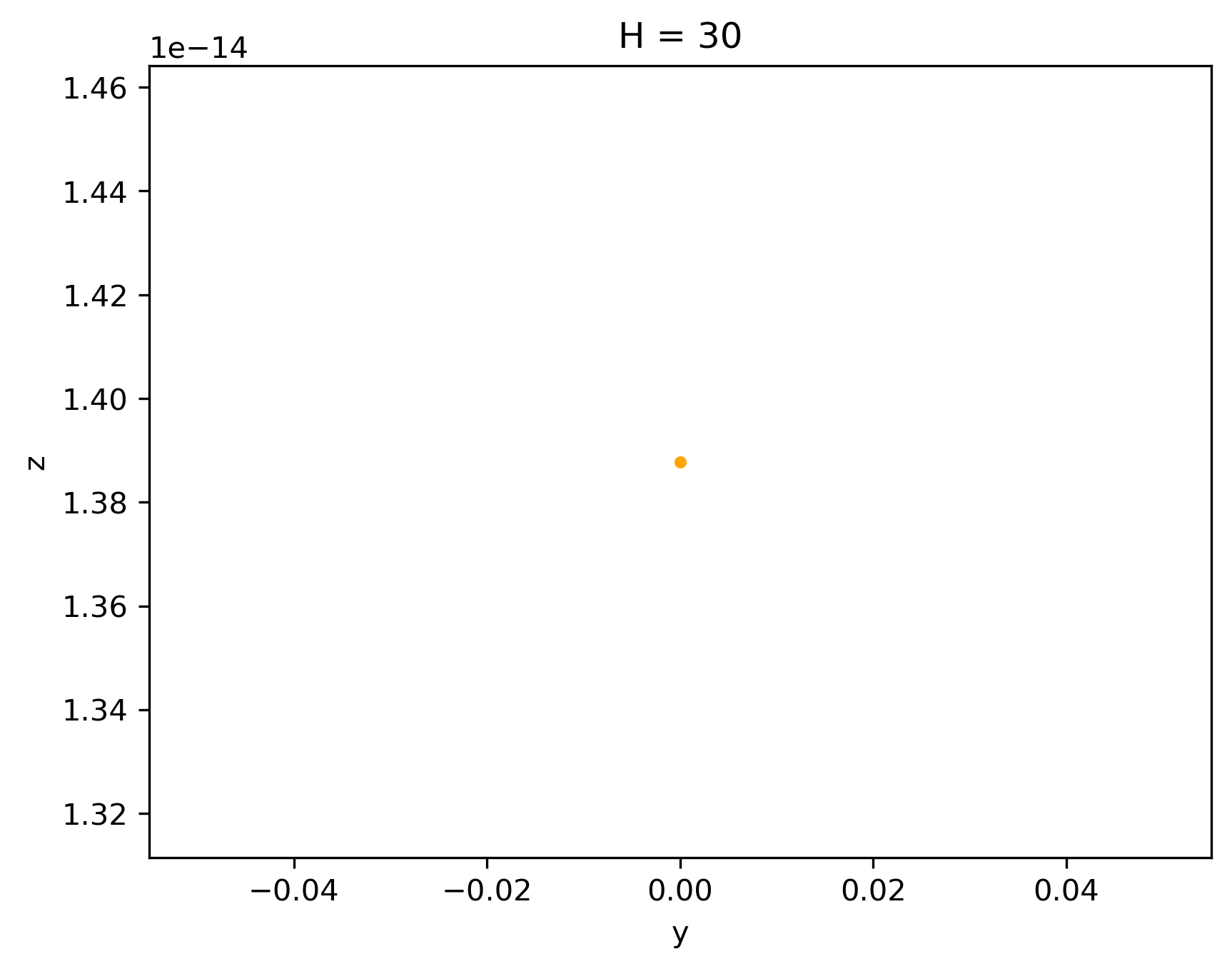


Figure 7 сечение тора плоскость х = 30

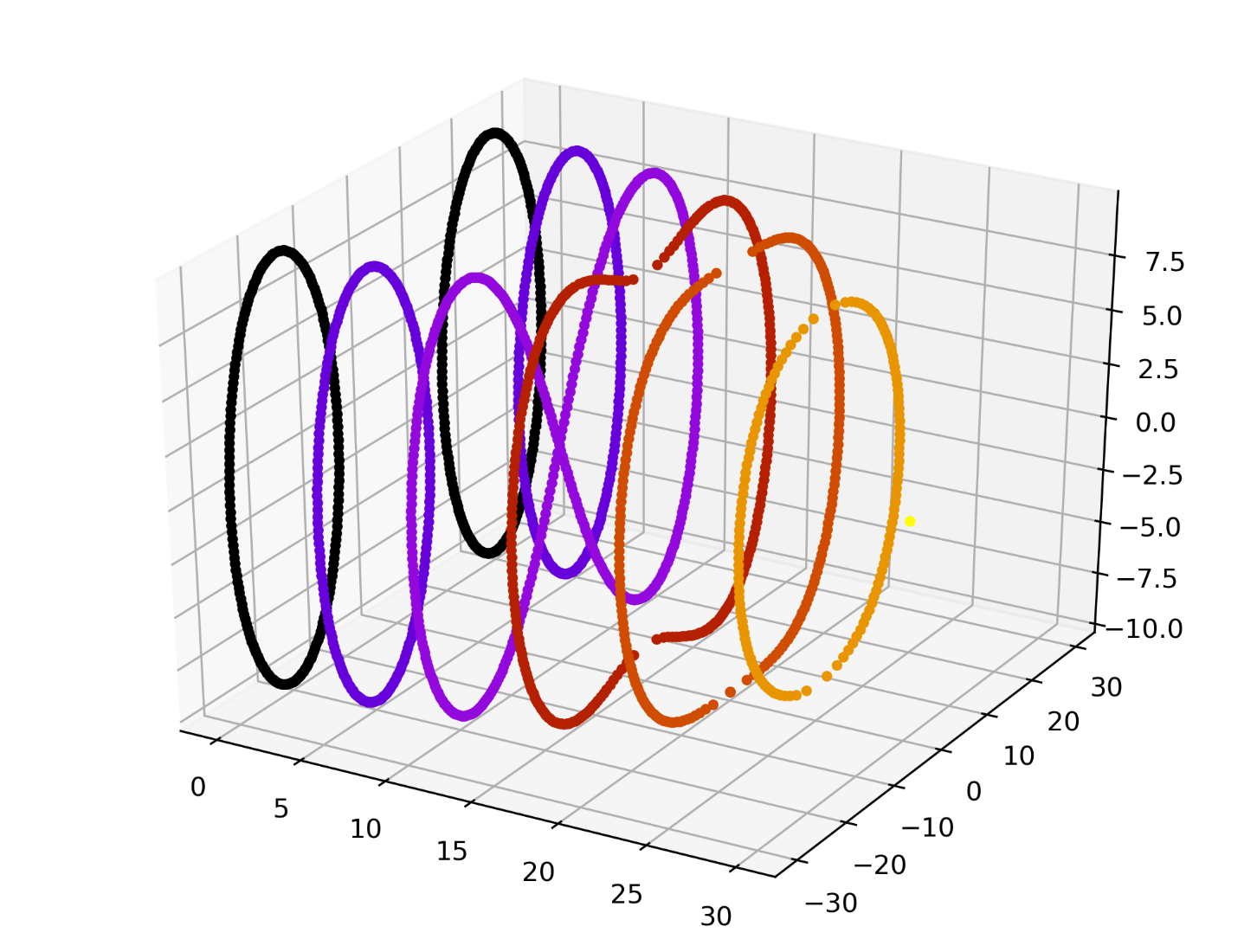


Figure 7 все сечения на одном графике

### Сравнение сечений с леминискатой Бернулли

Рассматривается тор со следующими характеристиками:

R = 20 – радиус вращения

r = 5 –радиус образующей

Параметр лемнискаты c = 20

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

Для их сравнения воспользуемся расстоянием Фреше

Расстояние Фреше = 5.3732

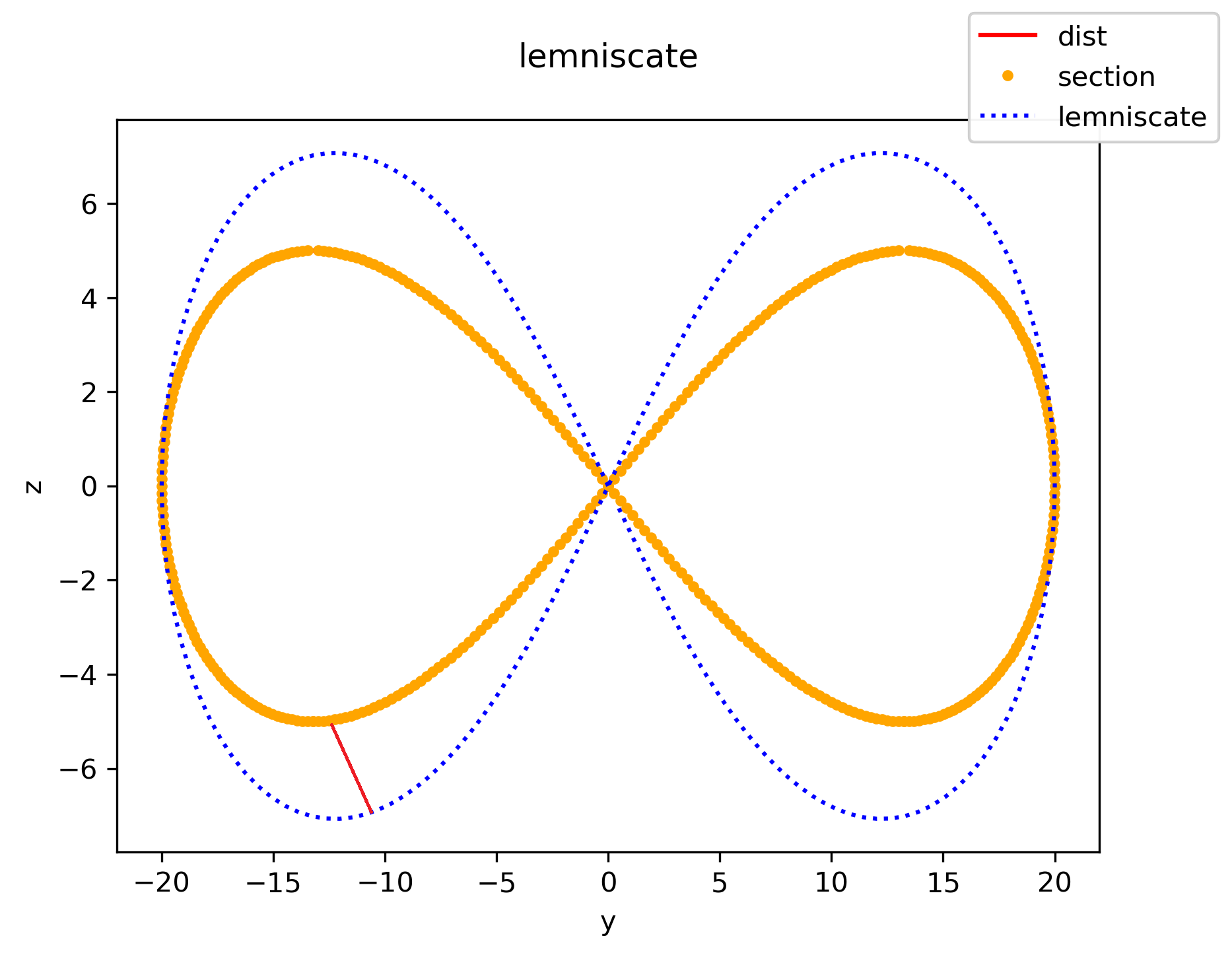


Figure 8 расстояние Фреше между лемнискатой и сечением тора x = 15

Путем подбора параметров тора и лемнискаты добьёмся их практического совпадения:

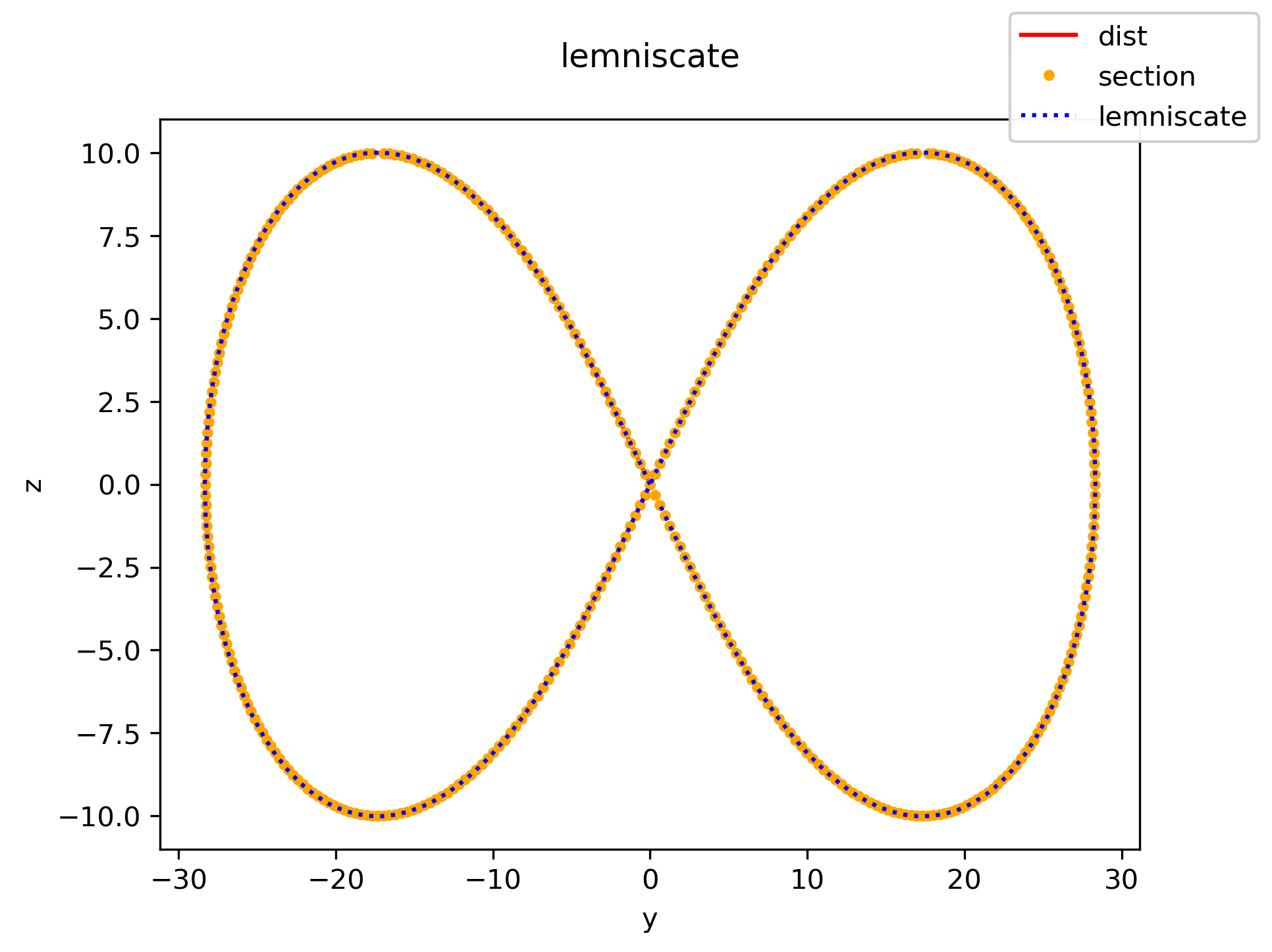
R = 20 – радиус вращения

r = 10 –радиус образующей

сечение плоскость x = 10

Параметр лемнискаты c = 28

Расстояние Фреше = 0.2381



## Обсуждение

По результатам видно, что общем случае сечение плоскость x=R-r действительно похоже на лемнискату Бернулли, но не совпадает с ним.

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")  
  
"""