Санкт-Петербургский политехнический университет им. Петра Великого
Институт прикладной математики и механики
Кафедра Прикладная математика

Лабораторная работа №2

Сечение тела вращения

по дисциплине: Вычислительные комплексы

Студент группы 3630102/60201 Митрофанова А.Г.

Преподаватель Баженов А.Н.

Санкт-Петербург
2019 год

Содержание

Список иллюстраций

1 Постановка задачи

Построить фигуру вращения, тор, как дискретный набор точек в трехмерном пространстве. Построить его сечение плоскостью $x = H$. Так как одно из сечений хорошо описывается лемнискатой Бернулли, построить леминискату и сравнить её с соответствующим сечением. Варьируя параметрами лемнискаты и тора, оптимизировать расстояние Фреше между ними.

2 Теория

Тор (тороид) - поверхность вращения, получаемая вращением образующей окружности вокргу оси, лежащей в плоскости этой окружности и не пересекающей её.
Сечение задаётся в плоскости $X0Z$. Тор получаем путём вращения всех точек по окружности вокруг Z. Сечение представляет собой дискретный набор точек, каждая из них движется по окружности. Для каждой точки найдём пересечение её окружности и плоскости $x = H$. Получается, что надо решить систему

$$\begin{cases} x^2 + y^2 = R^2 \\ x = H \\ z = z' \end{cases} \tag{1}$$

Решением данной системы будет $(H, \pm\sqrt{R^2 - H^2}, z')$. Если подкоренное выражение меньше нуля, то пересечения нет.
Представляет интерес сравнение аналитических кривых и алгоритмически полученных сечений. Для построения леминискату Бернулли удобно задать параметрически:

$$\begin{cases} x = \frac{c\sqrt{2}(p+p^3)}{1+p^4} \\ y = \frac{c\sqrt{2}(p-p^3)}{1+p^4} \\ p = \tan\alpha, \qquad \alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}] \end{cases} \tag{2}$$

3 Реализация

Реализация осуществлена на языке программирования Python в среде разработке JetBrains PyCharm. Были использованы две бибилотеки:

- NumPy – для работы с числами и массивами
- matplotlib – для построения двумерных графиков
- mpl_toolkits.mplot3d - для построения трехмерных графиков

4 Результаты

4.1 Построение сечений

Рассмотрим тор с параметрами $z = 0, R = 20, r = 10$, где z – ось, R – радиус вращения, r – радиус образующей.
Посмотрим, как выглядят сечения тора плоскостями $x = 0, 5, 10, 15, 20, 25, 30$.

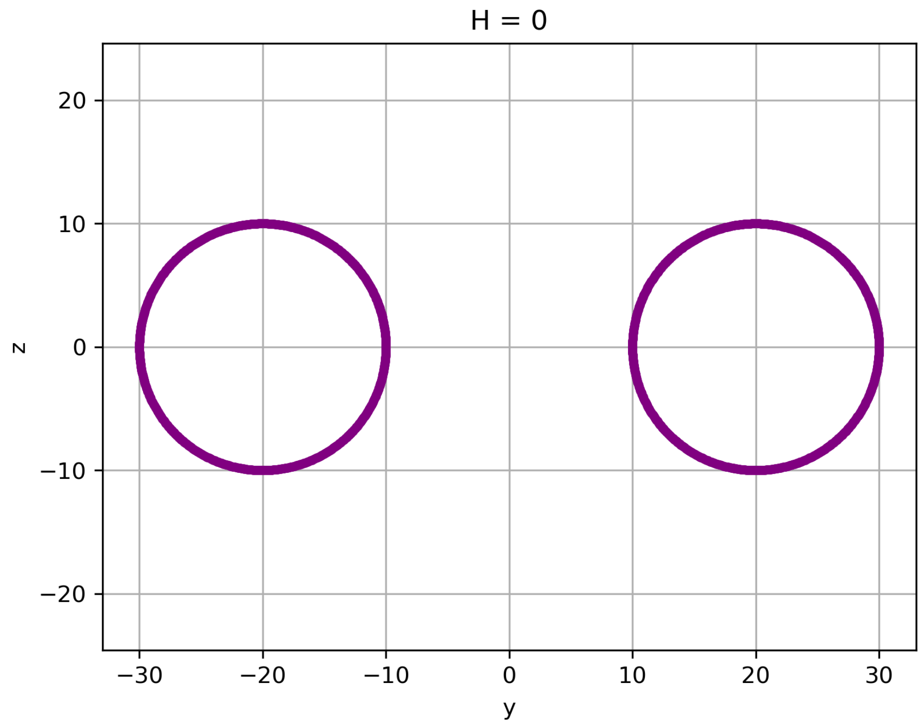

Рис. 1: Сечение тора плоскостью $x = 0$

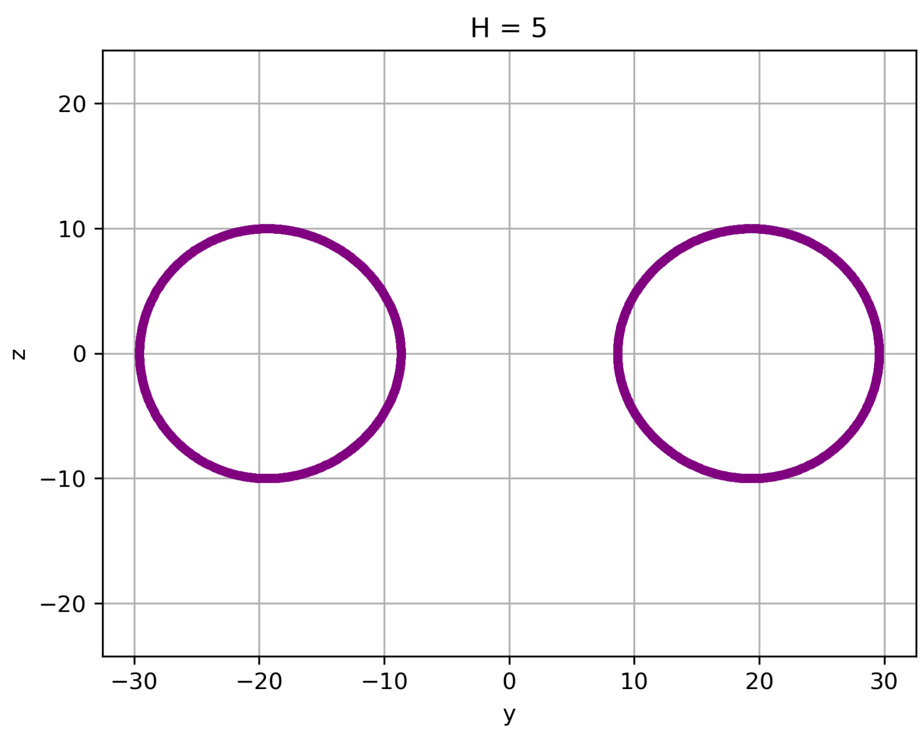

Рис. 2: Сечение тора плоскостью $x = 5$

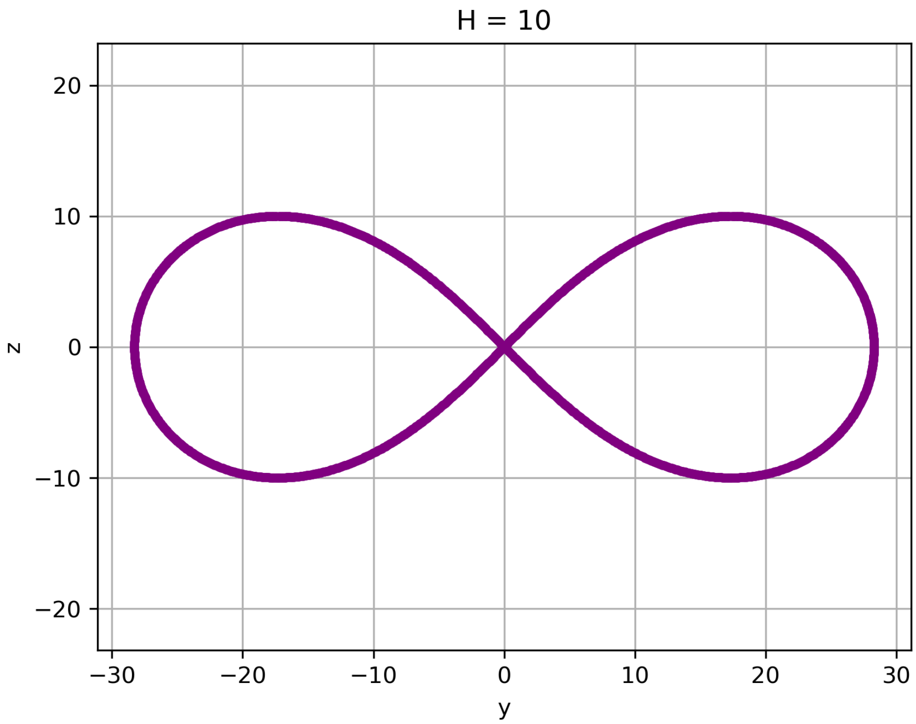

Рис. 3: Сечение тора плоскостью $x = 10$

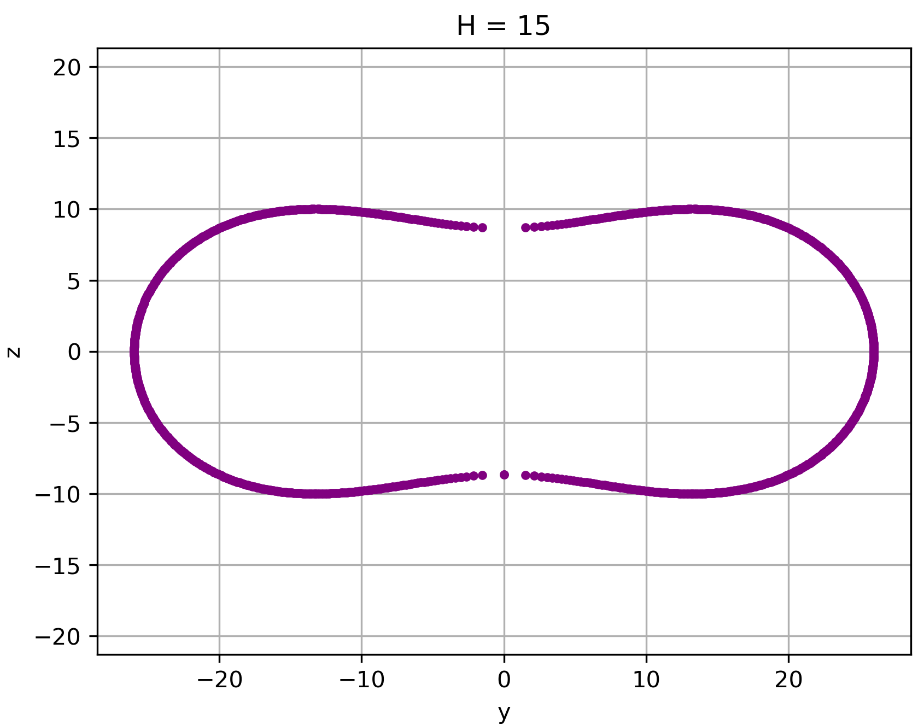

Рис. 4: Сечение тора плоскостью $x = 15$

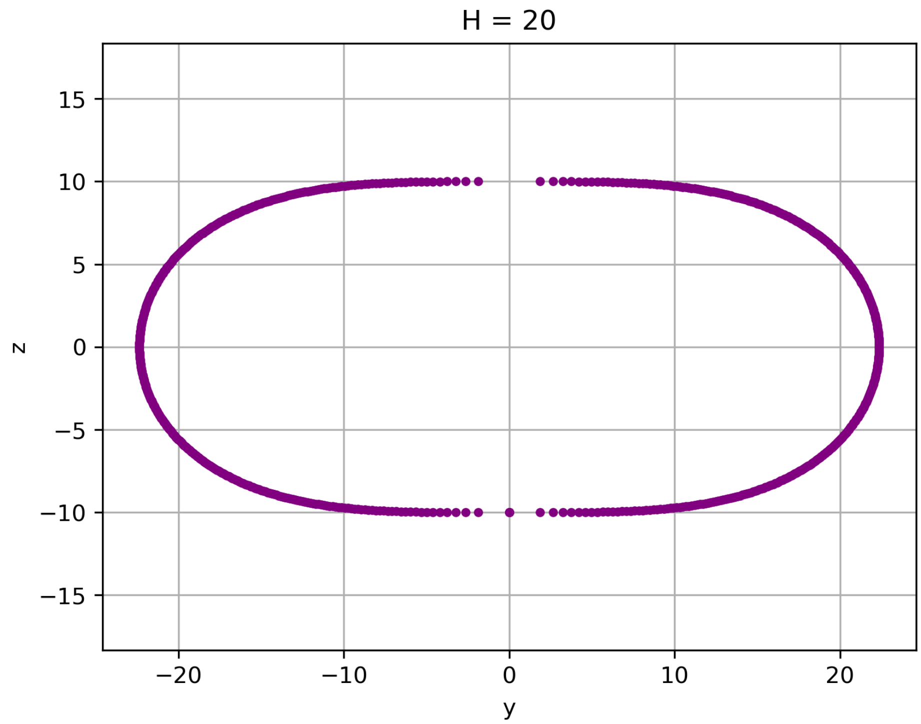

Рис. 5: Сечение тора плоскостью $x = 20$

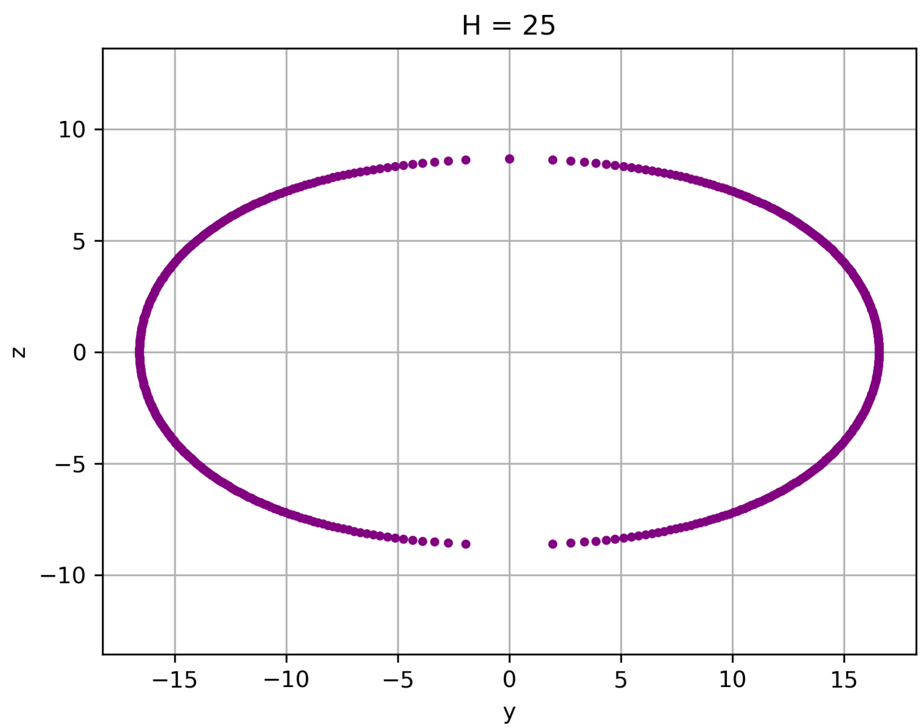

Рис. 6: Сечение тора плоскостью $x = 25$

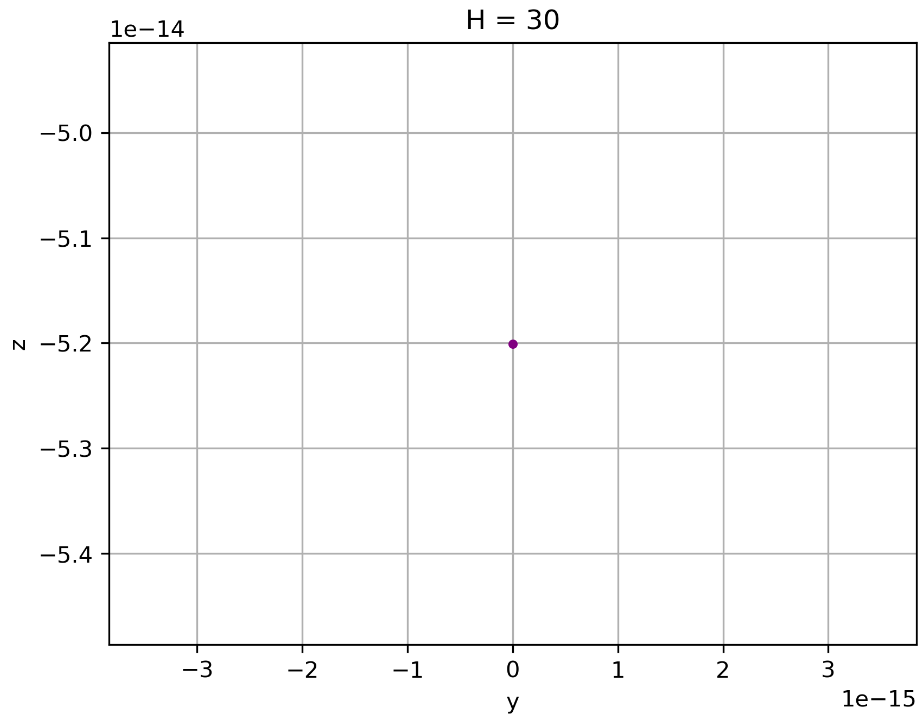

Рис. 7: Сечение тора плоскостью $x = 30$

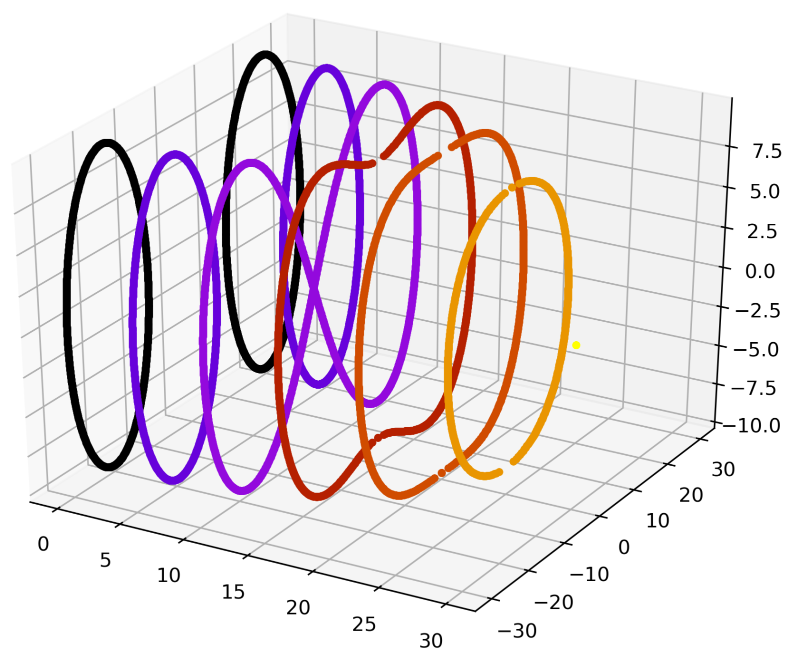

Рис. 8: Сечения тора плоскостями $x = 0$, $x = 5$, $x = 10$, $x = 15$, $x = 20$, $x = 25$, $x = 30$

Видим, что при $x = 10$ сечение тора очень похоже на леминискату Бернулли.

4.2 Сравнение сечений с леминискатой Бернулли

Рассмотрим тор с параметрами $z = 0, R = 20, r = 5$, где z – ось, R – радиус вращения, r – радиус образующей.
Посмотрим, как выглядит сечение тора плоскостью $x = R - r = 15$ и сравним его с леминискатой Бернулли с параметром $c = 20$.

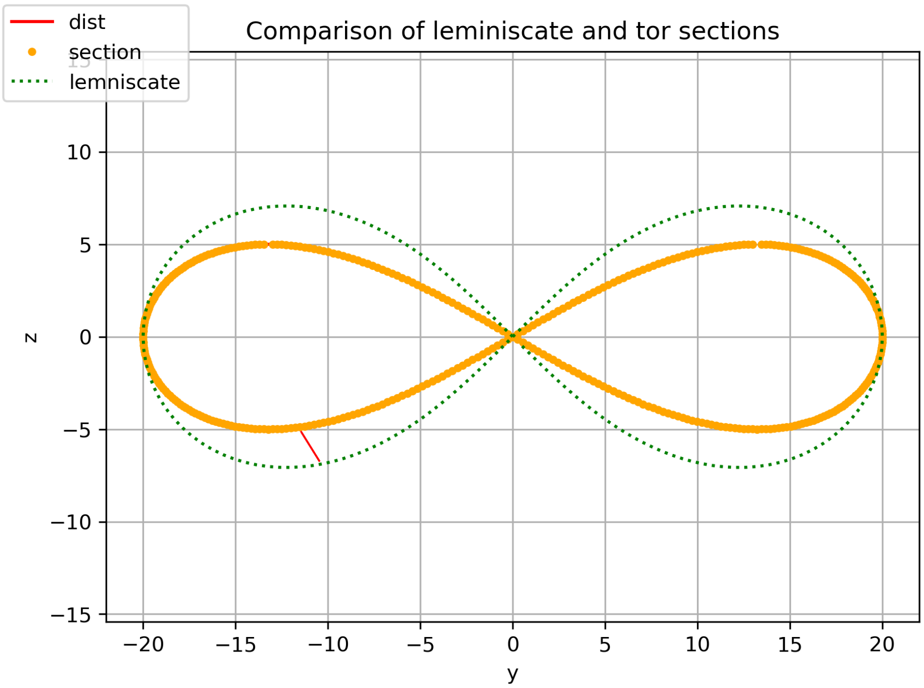

Рис. 9: Сравнение с леминискатой Бернулли

Видим, что они не совпадают. Воспользуемся расстоянием Фреше для сравнения сечения и леминискаты. Получаем $d = 5.3733$.
Подберем парметры тора и леминискаты таким образом, чтобы сечение тора практически совпадало с леминискатой.
Получаем тор из предыдущего пункта с параметрами $z = 0, R = 20, r = 10$, где z – ось, R – радиус вращения, r – радиус образующей. Берем сечение тора плоскостью $x = R - r = 10$ и леминискату Бернулли с параметром $c = 28$.
Воспользуемся расстоянием Фреше для сравнения сечения и леминискаты. Получаем $d = 0.2381$.

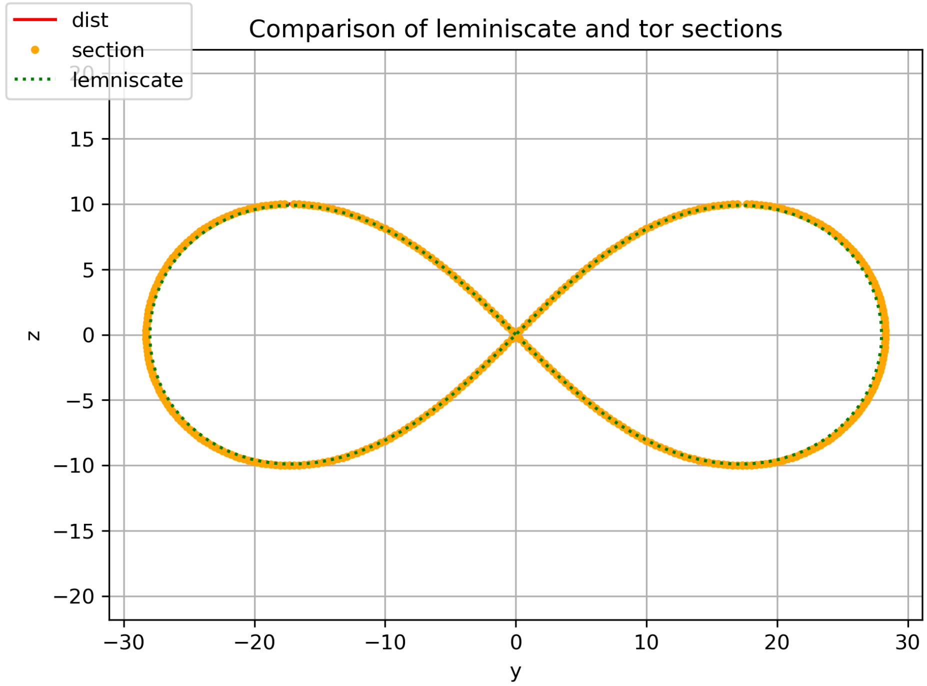

Рис. 10: Сравнение с леминискатой Бернулли

5 Обсуждение

В общем случаем сечение плоскостью $x = R - r$ похоже на леминискату Бернулли, но не совпадает полностью с ней.
Можно подобрать такие параметры тора и леминискаты, чтобы сечение практически совпадало с леминискатой по расстоянию Фреше. Оно будет в итоге мало, но никогда не будет равняться нулю, так как фигуры у нас заданы множеством точек (дискретизация). Для совпадения необходимо, чтобы радиус вращения тора R был в два раза больше радиуса образующей r.

6 Литература

Список литературы

[1] А.Н. Баженов, Лабораторный практикум. Методический материал. «Вычислительные комплексы» [Электронный ресурс, облачное хранилище]. Режим доступа: https://cloud.mail.ru/public/4ra6/5wwqBzMBC/LabPractics.pdf (дата обращения: сентябрь-декабрь, 2019 г.)

7 Приложение

7.1 section.py

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

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

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

    return left, right


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
        # distance from the center of the forming circle to the axis of rotation
        self.R_ = r_rotate
        # radius of the forming circle
        self.r_ = r_circle
        self.points_ = []

    def generate_points(self, num_points):
        self.points_ = []
        step = np.pi / num_points
        fi = - np.pi / 2

        for i in range(0, num_points):
            dr = self.r_ * np.cos(fi)
            point = TorPoint(self.r_ * np.sin(fi), 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)
```

```python
    point4 = Point(x_plane, - np.sqrt(r2 ** 2 - x_plane ** 2), z)

    return point1, point2, point3, point4


def find_intersection_tor_plane(tor, plane):
    tor_points = tor.get_tor_points()

    left1, left2, left3, left4 = [], [], [], []
    right1, right2, right3, right4 = [], [], [], []

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

    for point in tor_points:
        tmp = points_on_plane(point, 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()

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

    return left, right


def plot_3d(plane_cut, filename):
    fig = pylab.figure()
    axes = Axes3D(fig)
    number = len(plane_cut)
    cmap = plt.get_cmap('gnuplot')
    colors = [cmap(i) for i in np.linspace(0, 1, number)]

    i = 0
    for points_arr in plane_cut:
```

```
        for points in points_arr:
            x, y, z = [], [], []
            for elem in points:
                x.append(elem.x_)
                y.append(elem.y_)
                z.append(elem.z_)

            axes.plot(x, y, z, ".", color=colors[i])

        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='all')
    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_)
        plt.axis('equal')
        plt.grid(True)
        ax.plot(y, z, ".", color="purple")

    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width, box.height])
    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 = find_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, "tor_all_sections.png")


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])
        plt.axis('equal')
        plt.grid(True)
        ax.plot(x, y, ":", label="lemniscate", color=color)


def process_lemniscate(tor, plane, c):
    left_lemn, right_lemn = lemniscate(c)
    left_sec, right_sec = find_intersection_tor_plane(tor, plane)
    points_arr = [left_sec, right_sec]

    fig, ax = plt.subplots(nrows=1, ncols=1, sharey='all')
    ax.set_xlabel("y")
    ax.set_ylabel("z")
```

```
    ax.set_title("  omparison   of leminiscate and tor sections")
    plt.axis('equal')
    plt.grid(True)

    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_)
            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(left_lemn + right_lemn, ax, "green")

    solver1 = frechet.Frechet(right_lemn, data_arr[1])
    solver2 = frechet.Frechet(left_lemn, data_arr[0])

    dist1, i1, j1 = solver1.frechet_distance()
    dist2, i2, j2 = solver2.frechet_distance()
    print("dist1 = ", dist1, "i1 = ", i1, "j1 = ", j1)
    print("dist2 = ", dist2, "i2 = ", i2, "j2 = ", j2)

    fig.legend(loc='upper left')
    fig.savefig("comparison.png", dpi=300, format='png', bbox_inches='tight')

    fig.show()
    plt.close(fig)
```

7.2 frechet.py

```
import numpy as np
import matplotlib.pyplot as plt


class Frechet:
    def __init__(self, P, Q):
        self.P = np.array(P)
        self.Q = np.array(Q)

        self.p = len(P)
        self.q = len(Q)

        self.ca = np.full((self.p, self.q), -1.0)

    def frechet_distance(self):
        dist, i, j = self.c(self.p - 1, self.q - 1)
        self.plot(i, j, dist)
        return dist, i, j

    def c(self, i, j):
        n_i = i
        n_j = j
        d = np.linalg.norm(self.P[i] - self.Q[j])
        if self.ca[i][j] > -1:
            return self.ca[i][j], n_i, n_j
        elif i == 0 and j == 0:
            self.ca[i][j] = d
        elif i > 0 and j == 0:
            self.ca[i][j], n_i, n_j = max(
                self.c(i - 1, 0), (d, i, 0)
```

```
            )
        elif i == 0 and j > 0:
            self.ca[i][j], n_i, n_j = max(
                self.c(0, j - 1), (d, 0, j)
            )
        elif i > 0 and j > 0:
            self.ca[i][j], n_i, n_j = max(min(
                self.c(i - 1, j), self.c(i - 1, j - 1), self.c(i, j - 1)),
                (d, i, j)
            )
        else:
            self.ca[i][j] = float('inf')

        return self.ca[i][j], n_i, n_j

    def plot(self, i, j, d):
        plt.figure()
        plt.plot(self.P[:, 0], self.P[:, 1], color='blue')
        plt.plot(self.Q[:, 0], self.Q[:, 1], color='orange')
        plt.plot([self.P[i][0], self.Q[j][0]], [self.P[i][1], self.Q[j][1]], color='red')
        plt.legend(['P', 'Q', 'Frechet distance = %.3f' % d])
        plt.title('Frechet distance')
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.axis('equal')
        plt.grid(True)

        plt.savefig("Frechet_dist%d.png" % d, dpi=500, format='png')
        plt.show()
```

7.3 main.py

```
import section

if __name__ == "__main__":
    R = 20
    r = 10
    z = 0
    c = 28
    n = 100

    tor = section.Tor(z, R, r)
    tor.generate_points(n)

    H = [0, 5, 10, 15, 20, 25, 30]
    section.plane_research(tor, H)
    section.process_lemniscate(tor, R - r, c)
```