**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра алгоритмической математики**

# КУРСОВАЯ РАБОТА

**по дисциплине «Дифференциальные уравнения»**
**Тема: Остывание предмета в комнате**

Студенты гр. 8382 Кобенко В.П.
Черницын П.А.

Преподаватель Павлов Д.А.

Санкт-Петербург
2021

**ЗАДАНИЕ**
**НА КУРСОВУЮ РАБОТУ**

Студент Кобенко В.П.
Студент Черницын П.А.
Группа 8382
Тема работы: Остывание предмета в комнате

Исходные данные:
Остывание предмета в комнате

Содержание пояснительной записки:
«Содержание», «Введение», «Закон Ньютона-Рихмана», «Прямой метод Ньютона», «Обратный метод Ньютона», «Метод Рунге-Кутты-Фельберга 4-5-го порядка», «Метод Хойна», «Метод Адамса», «Графический интерфейс», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 20 страниц.

Дата выдачи задания: 09.05.2021

Дата сдачи курсовой работы: 14.06.2021

Дата защиты курсовой работы: 14.06.2021

| | | |
|---|---|---|
| Студенты | | Кобенко В.П.<br>Черницын П.А. |
| Преподаватель | | Павлов Д.А. |

## АННОТАЦИЯ

В курсовой работе рассмотрена задача остывания предмета в комнате. Для этого использовался закон Ньютона – Рихмана, его дифференциальная формулировка. Для решения поставленной задачи было использовано несколько методов: «Прямой метод Ньютона», «Обратный метод Ньютона», «Метод Рунге-Кутты-Фельберга 4-5-го порядка», «Метод Хойна», «Метод Адамса». Результаты решения данного уравнения были представлены в виде графиков в графическом интерфейсе.



## SUMMARY

In the course work, the problem of cooling an object in a room is considered. For this, the Newton-Richman law, its differential formulation, was used. To solve the problem, several methods were used: "Forward Newton's method", "Backward Newton's method", "Runge-Kutta-Felberg method of the 4-5th order", "Heun's method", "Adams method". The results of solving this equation were presented in the form of graphs in the graphical interface.

# СОДЕРЖАНИЕ

## Введение

Дифференциальное уравнение является одним из фундаментальных понятий математики, широко применяемое в различных областях современных наук. Оно также применимо в физических процессах, один из которых рассматривается в данной курсовой работе. Остывание предмета в комнате является этим процессом. Были использованы методы интегрирования дифференциальных уравнений динамических систем для решения закона Ньютона-Рихмана, такие как: «Прямой метод Ньютона», «Обратный метод Ньютона», «Метод Рунге-Кутты-Фельберга 4-5-го порядка», «Метод Хойна», «Метод Адамса».

## Закон Ньютона-Рихмана

Закон Ньютона-Рихмана — эмпирическая закономерность, выражающая тепловой поток между разными телами через температурный напор. Он будет использоваться для подсчета скорости остывания тела. Дифференциальная формулировка выглядит так:

$$\frac{dT}{dt} = +k * (T - T0), \text{ при k} < 0$$

$$\frac{dT}{dt} = -k * (T - T0), \text{ при k} > 0$$

Где T – температура предмета, t – время, T0 – температура окружения(в нашем случае – это температура комнаты), k – коэффициент(зависящая от объекта).

Так как мы рассматриваем задачу остывания тела до температуры окружения, то будем использовать эту формулу:

$$\frac{dT}{dt} = -k * (T - T0)$$

Умножим уравнение на $\frac{dt}{(T-T0)}$ и получим:

$$\frac{dT}{(T - T0)} = -k * dt$$

Проинтегрируем обе части уравнения:

$$\int \frac{1}{(T - T0)} dT = -k * \int dt$$

Получаем:

$$\ln|T - T0| = -kt + const$$

Избавимся от логарифма и получим:

$$|T - T0| = C * \exp(-kt)$$

Раскроем модуль:

$$T - T0 = \pm C * \exp(-kt)$$

$$T(t) = T0 + C * \exp(-kt)$$

Пусть *T(0) = T'* будет начальным условием(т.е температурой предмета), тогда формула примет вид:

$$T = T0 + (T' - T0) * \exp(-kt)$$

## Прямой метод Эйлера

Пусть дана система дифференциальных уравнений с непрерывным временем:

$$\dot{x} = f(x,t), x(t_0) = x_0,$$

последовательностью точек $x_0, x_1, \ldots$ в соответствующие моменты времени $t_0, t_1, \ldots$ Значения точек должны удовлетворять приближенному равенству:

$$x_k \approx \psi_t(x_0, t_0)$$

Если специально не оговорено иное, то предполагается, что моменты времени выбираются через равные интервалы с величиной шага $h>0$, то есть

$$t_{k+1} = t_0 + kh, \quad k = 0, 1, \ldots$$

Аппроксимируем производную в момент времени $t_k$ соотношением

$$\dot{x}(t_k) \approx (x_{k-1} - x_k)/h$$

При такой аппроксимации уравнение примет вид:

$$x_{k+1} = x_k + hf(x_k, t_k),$$

Эта формула известна как прямой метод Эйлера.

В нашем коде этот метод был реализован так:

```
for i in range(x_len):
    F_x_t = mf.myFunc(y, self.env_temp)

    for j in range(y_len):
        y[j] = y[j] + self.h*F_x_t[j]

    x += self.h
    self.f.write(str(x) + ' ')

    for r in range(len(y)):
        y_res = np.append(y_res, y[r])
```

F_x_t – функция f(x, t), self.h – это длина шага по x, y[j] – температура на текущем шаге, my_func.py выглядит так:

```
def myFunc(y, env_temp):
    dy = np.zeros((len(y)))
    # dy[0] = 3*(1+x) - y[0]
    dy[0] = 1/25 * np.log(2) * (env_temp - y)
    return dy
```

,

Где env_temp – это температура окружения, а 1/25 * np.log(2) – k.

## Обратный метод Эйлера

Обратный метод Эйлера подобен прямому, но есть одно отличие в аппроксимации для производной:

$$\dot{x}(t_k) \approx \frac{x_k - x_{k-1}}{h}$$

Такая аппроксимация дает формулу обратного метода Эйлера:

$$x_{k+1} = x_k + hf(x_{k+1}, t_{k+1})$$

Обратный метод Эйлера - это пример неявного алгоритма интегрирования, где $x_{k+1}$ является функцией от самой себя. И напротив, прямой метод Эйлера представляет собой явный алгоритм. В неявных алгоритмах для определения $x_{k+1}$ требуются дополнительные вычисления, но они по сравнению с аналогичными прямыми алгоритмами более устойчивы и дают более высокую точность вычислений.

В нашем коде этот метод был реализован так:

```
for i in range(x_len):
    F_x_t = mf.myFunc(y, self.env_temp)/(1+self.h)
    # print(">> ", x, h, y, F_x_t)

    for j in range(y_len):
        y[j] = y[j] + self.h*F_x_t[j]

    x += self.h
    self.f.write(str(x) + ' ')

    for r in range(len(y)):
        y_res = np.append(y_res, y[r])
```

Где F_x_t – функция f(x, t), self.h – это длина шага по x, y[j] – температура на текущем шаге, self.env_temp – температура окружения.

## Метод Рунге-Кутты-Фельберга 4-5 порядка

Метод Рунге-Кутты-Фельберга 4-5 порядка – адаптивный метод, в котором каждый шаг требует использования шести следующих значений:

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{5}h, y_i + \frac{1}{5}hk_1\right)$$

$$k_3 = f\left(x_i + \frac{3}{10}h, y_i + \frac{3}{40}hk_1 + \frac{9}{40}hk_2\right)$$

$$k_4 = f\left(x_i + \frac{3}{5}h, y_i + \frac{3}{10}hk_1 - \frac{9}{10}hk_2 + \frac{6}{5}hk_3\right)$$

$$k_5 = f\left(x_i + h, y_i - \frac{11}{54}hk_1 + \frac{5}{2}hk_2 - \frac{70}{27}hk_3 + \frac{35}{27}hk_4\right)$$

$$k_6 = f\left(x_i + \frac{7}{8}h, y_i + \frac{1631}{55296}hk_1 + \frac{175}{512}hk_2 + \frac{575}{13824}hk_3 + \frac{44275}{110592}hk_4 + \frac{253}{4096}hk_5\right)$$

Аппроксимация для этого метода выглядит так:

$$y_{i+1} = y_i + h\left(\frac{37}{378}k_1 + \frac{250}{621}k_3 + \frac{125}{594}k_4 + \frac{512}{1771}k_6\right)$$

Этот метод был реализован таким образом:

```
for i in range(x_len):
    k1 = mf.myFunc(y, self.env_temp)

    yp2 = y + k1*(h/5)

    k2 = mf.myFunc(yp2, self.env_temp)

    yp3 = y + k1*(3*h/40) + k2*(9*h/40)

    k3 = mf.myFunc(yp3, self.env_temp)

    yp4 = y + k1*(3*h/10) - k2*(9*h/10) + k3*(6*h/5)

    k4 = mf.myFunc(yp4, self.env_temp)

    yp5 = y - k1*(11*h/54) + k2*(5*h/2) - k3*(70*h/27) + k4*(35*h/27)

    k5 = mf.myFunc(yp5, self.env_temp)

    yp6 = y + k1*(1631*h/55296) + k2*(175*h/512) + k3*(575*h/13824) + k4*(44275*h/110592) + k5*(253*h/4096)

    k6 = mf.myFunc(yp6, self.env_temp)

    for j in range(y_len):
        y[j] = y[j] + h*(37*k1[j]/378 + 250*k3[j]/621 + 125*k4[j]/594 + 512*k6[j]/1771)
```

## Метод Хойна

В математике и вычислительной науке, Метод Хойна(Разностная схема Хойна) может относиться к улучшенному или модифицированному методу Эйлера (то есть, явному правилу трапеции ) или аналогичному двух- этап Метод Рунге – Кутта . Он назван в честь Карла Хойна и представляет собой числовую процедуру для решения обыкновенных дифференциальных уравнений с заданным начальным значением . Оба варианта можно рассматривать как расширение метода Эйлера до двухэтапных методов Рунге – Кутты второго порядка.

Разностную схему Хойна (вернее рекуррентный переход от xi–1 к xi) часто записывают в виде двух полушагов, поэтому на обычно называется схемой предиктор-корректор: на первом полушаге приближенное решение "предсказывается" (от англ. to predict — предсказывать) с первым порядком точности, а на втором — "корректируется" (от to correct — исправлять, корректировать) с целью повышения точности.

В работе метод был реализован так:

```
for i in range(x_len):
    y0prime = mf.myFunc(y, self.env_temp)

    k1 = y0prime * h

    ypredictor = y + k1

    y1prime = mf.myFunc(ypredictor, self.env_temp)

    for j in range(y_len):
        y[j] = y[j] + (h/2)*y0prime[j] + (h/2)*y1prime[j]

    x = x + h
    self.f.write(str(x) + ' ')

    for r in range(len(y)):
        y_res = np.append(y_res, y[r])
```

Где ypredictor – является предиктором, а корректором является выражение в цикле for j in range(y_len).

## Метод Адамса-Башфорта

Метод Адамса — конечноразностный многошаговый метод численного интегрирования обыкновенных дифференциальных уравнений первого порядка. В отличие от метода Рунге-Кутты использует для вычисления очередного значения искомого решения не одно, а несколько значений, которые уже вычислены в предыдущих точках.

Пусть дана система дифференциальных уравнений первого порядка

$$y' = f(x, y), y(x_0) = y_0,$$

для которой надо найти решение на сетке с постоянным шагом

$$x_n - x_0 = (n - 1)h$$

Формула метода Адамса для решения этой системы имеет вид:

$$y_{n+1} = y_n + h\sum_{\lambda=0}^{k} u_{-\lambda} f(x_{n-\lambda}, y_{n-\lambda}),$$

Явные методы Адамса — Башфорта:

$$y_{n+1} = y_n + hf(t_n, y_n),$$

$$y_{n+2} = y_{n+1} + h\left(\frac{3}{2}f(t_{n+1}, y_{n+1}) - \frac{1}{2}f(t_n, y_n)\right),$$

$$y_{n+3} = y_{n+2} + h\left(\frac{23}{12}f(t_{n+2}, y_{n+2}) - \frac{4}{3}f(t_{n+1}, y_{n+1}) + \frac{5}{12}f(t_n, y_n)\right),$$

$$y_{n+4} = y_{n+3} + h\left(\frac{55}{24}f(t_{n+3}, y_{n+3}) - \frac{59}{24}f(t_{n+2}, y_{n+2}) + \frac{37}{24}f(t_{n+1}, y_{n+1}) - \frac{3}{8}f(t_n, y_n)\right),$$

Для предварительного вычисления решения в k начальных точках был использован метод Рунге-Кутты 4-ого порядка. Сам же метод Адамса-Башфорта в работе выглядит так:

```
for i in range(3, dx):
    x00 = self.x[i]; x11 = self.x[i-1]; x22 = self.x[i-2]; x33 = self.x[i-3]; xpp = self.x[i]+self.h

    y00 = np.array([y[i]])
    y11 = np.array([y[i - 1]])
    y22 = np.array([y[i - 2]])
    y33 = np.array([y[i - 3]])

    y0prime = mf.myFunc(y00, self.env_temp)
    y1prime = mf.myFunc(y11, self.env_temp)
    y2prime = mf.myFunc(y22, self.env_temp)
    y3prime = mf.myFunc(y33, self.env_temp)

    ypredictor = y00 + (self.h/24)*(55*y0prime - 59*y1prime + 37*y2prime - 9*y3prime)
    ypp = mf.myFunc(ypredictor, self.env_temp)

    for j in range(y_len):
        yn[j] = y00[j] + (self.h/24)*(9*ypp[j] + 19*y0prime[j] - 5*y1prime[j] + y2prime[j])

    xs = self.x[i] + self.h
    xsol = np.append(xsol, xs)

    self.x = xsol

    for r in range(len(yn)):
        y_res = np.append(y_res, yn)
```

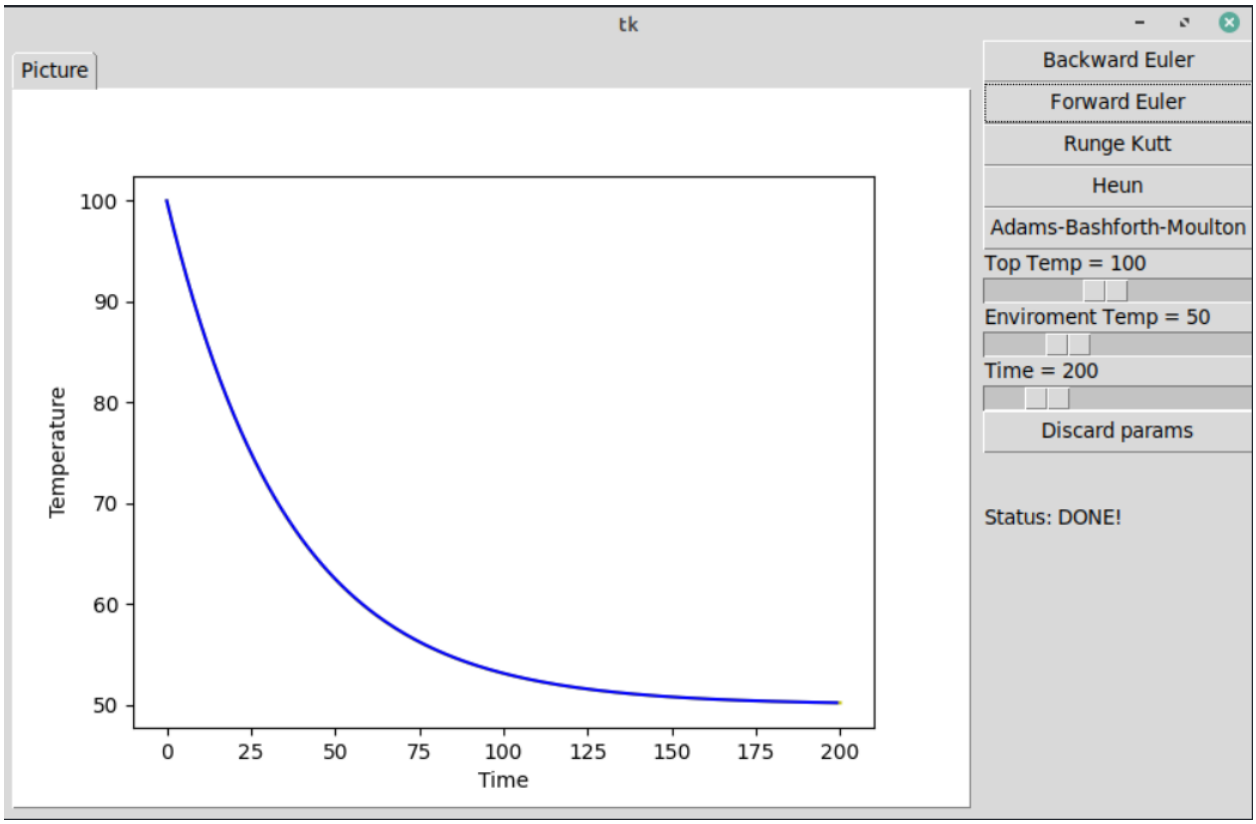
## GUI

Был реализован графический интерфейс на языке Python с помощью библиотеки PySimpleGUI. Код программы представлен в приложении А.

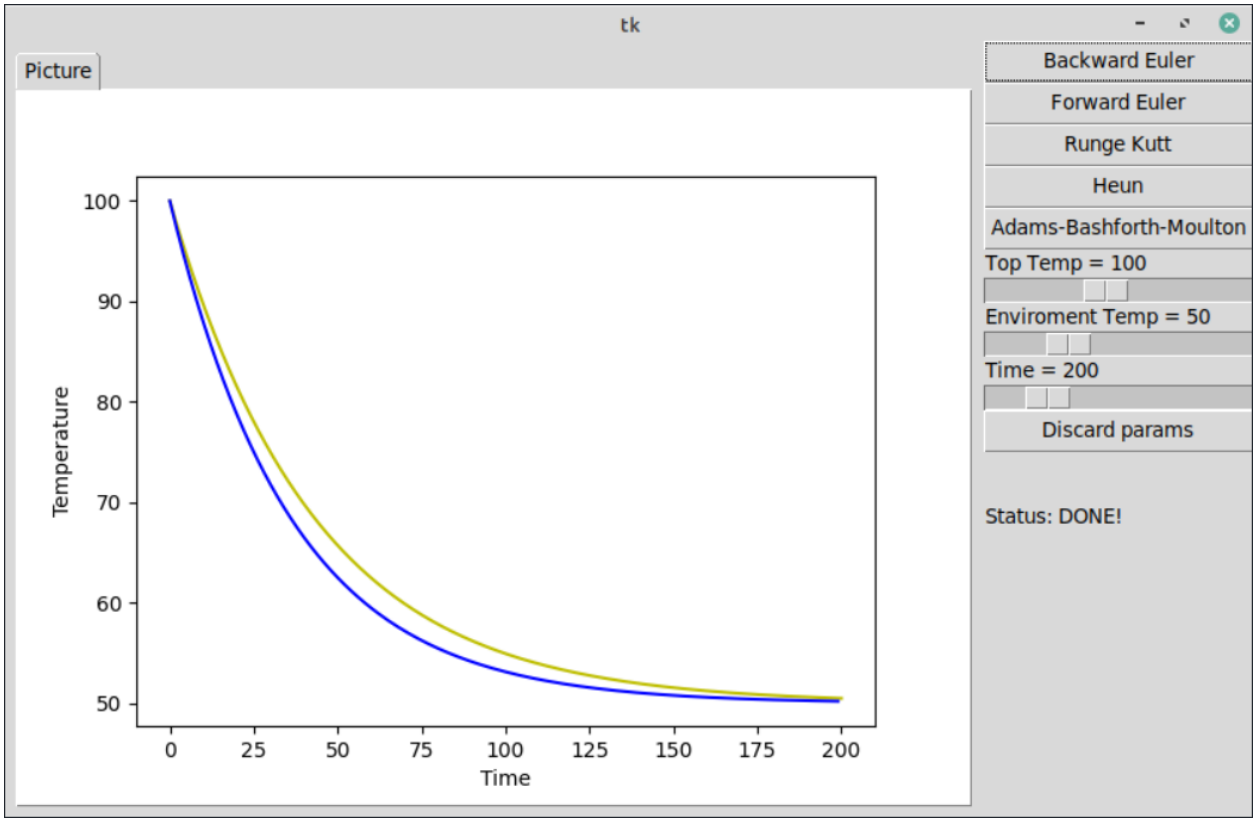
Интерфейс программы включает в себя 6 кнопок, 3 слайдера, окно для графиков и лэйбл с выводом информации об успешном/неуспешном запуске программы:

Первые пять кнопок в интерфейсе вызывают численные методы:

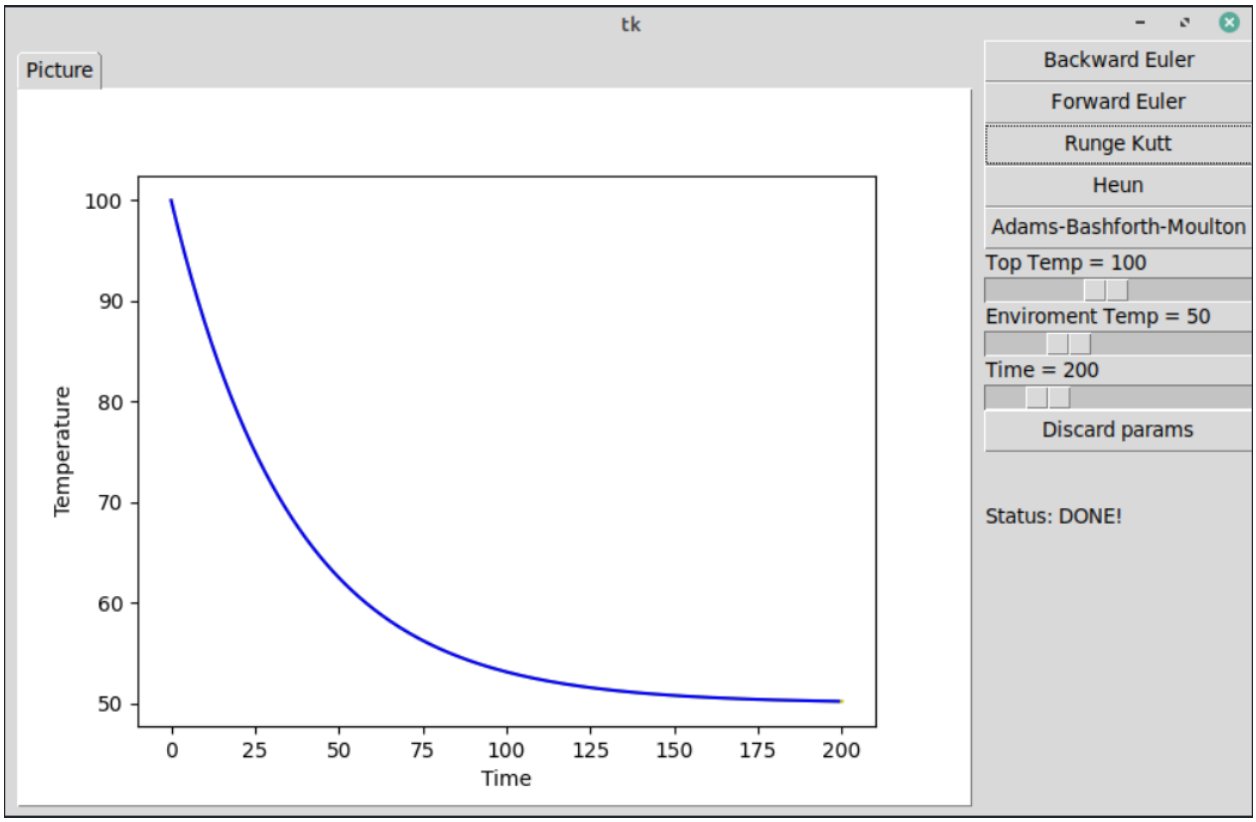
Forward Euler:



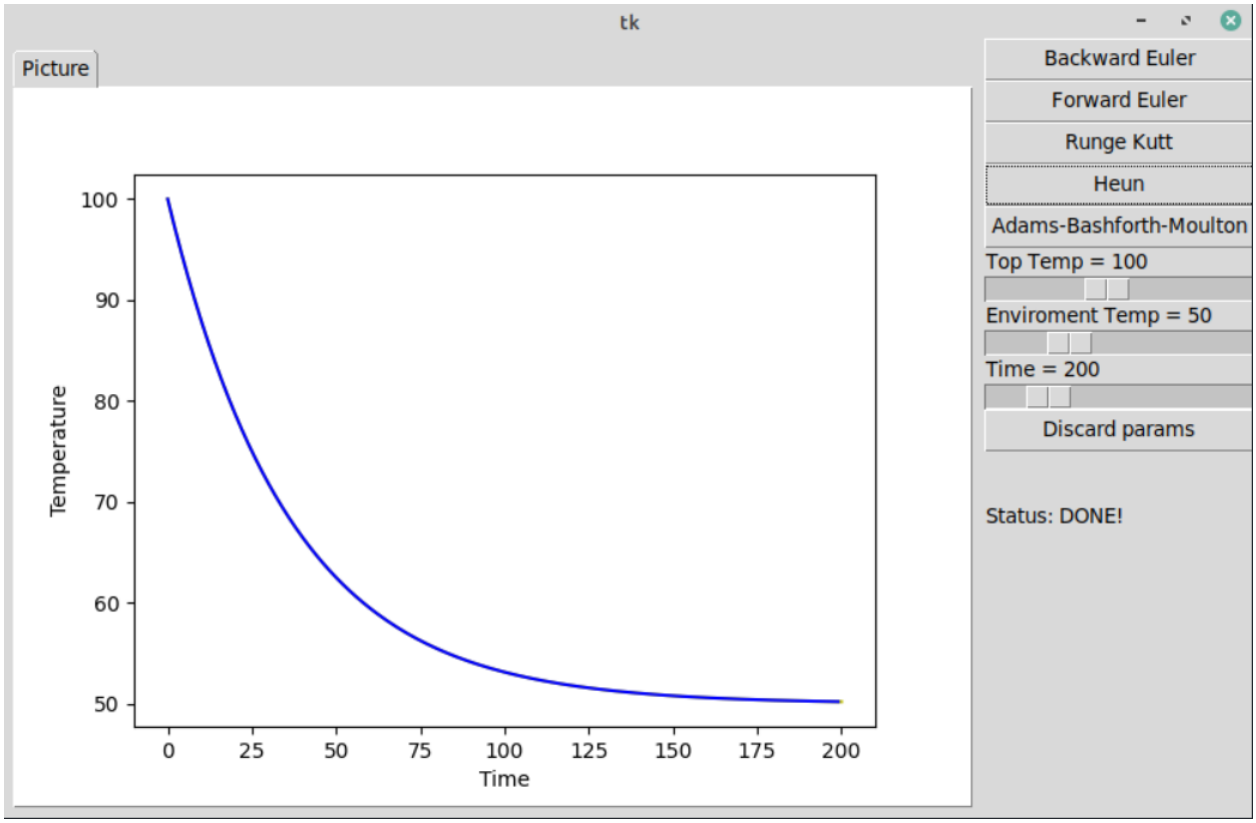
Backward Euler:



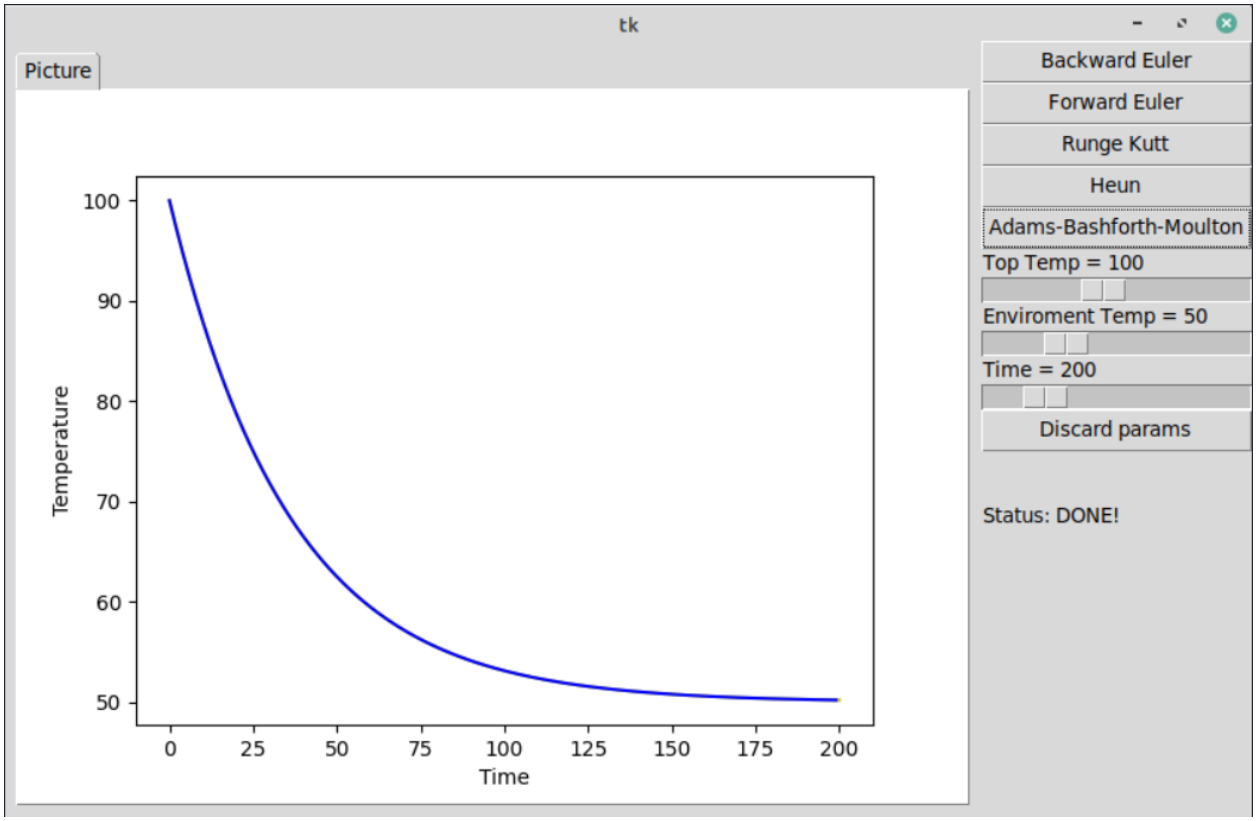

Runge Kutt:



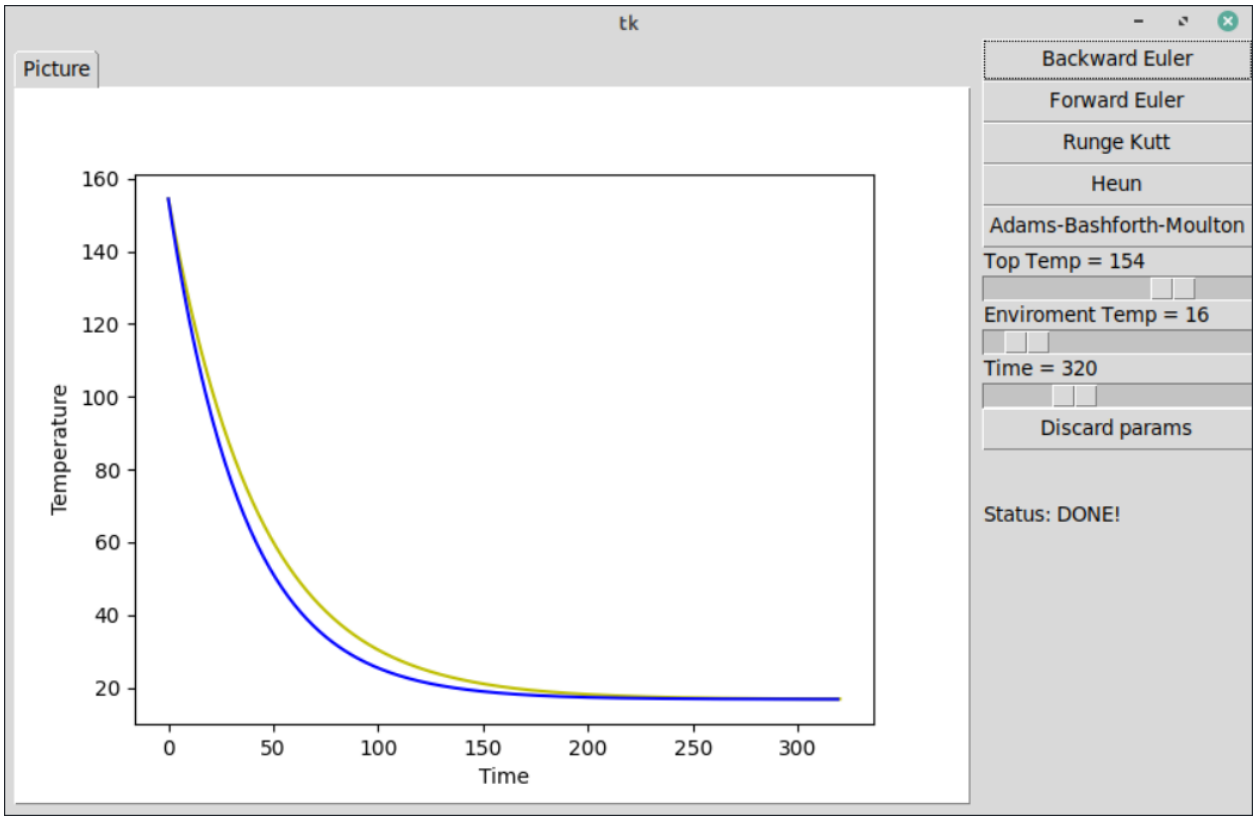
Heun:




Adams-Bashfourth:

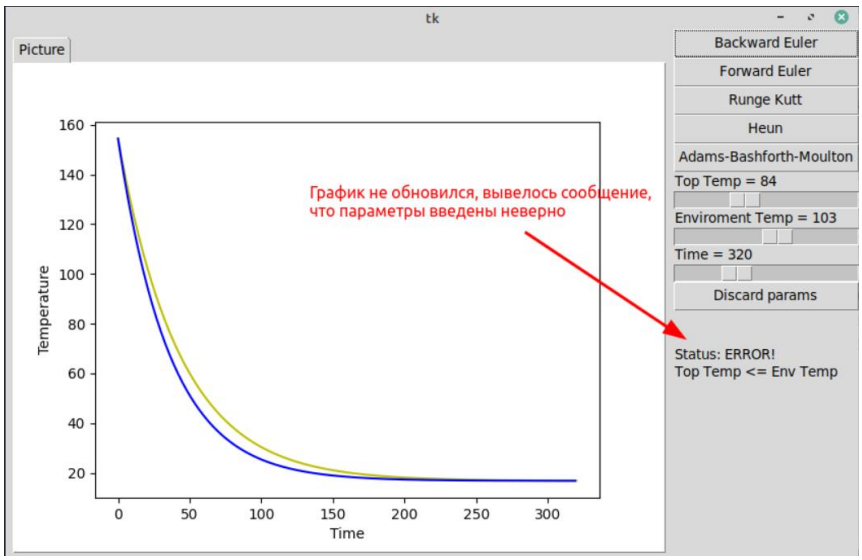


С помощью слайдеров можно менять условия поставленной задачи:




Top temp отвечает за температуру предмета, Enviroment Temp за температуру среды, Time – за время, кнопка Discard params – сбрасывает значения на исходные.

Если температура окружающей среды окажется больше, чем температура предмета, то выведется такой результат:



## Вывод

В курсовой работе был рассмотрен процесс остывания тела в комнате. Для решения задачи были использованы такие методы, как: «Прямой метод Ньютона», «Обратный метод Ньютона», «Метод Рунге-Кутты-Фельберга 4-5-го порядка», «Метод Хойна», «Метод Адамса». Было написано приложение на языке Python, решающее поставленную задачу данными методами. Графический интерфейс позволяет увидеть отличия между методами на графиках.

## Используемая литература


https://old.math.tsu.ru/EEResources/pdf/diff_equation.pdf
http://math.smith.edu/~callahan/cic/ch4.pdf
https://en.wikipedia.org/wiki/Newton%27s_law_of_cooling
https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%90%D0%B4%D0%B0%D0%BC%D1%81%D0%B0
http://w.ict.nsc.ru/books/textbooks/akhmerov/nm-ode_unicode/1-3.html
https://tftwiki.ru/wiki/Heun%27s_method
https://pysimplegui.readthedocs.io/en/latest/
https://www.python.org/

## ПРИЛОЖЕНИЕ А. Код программы

Файл main.py:

```
import tkinter as tk
import gui


root = tk.Tk()
gui.MainApplication(root)
root.mainloop()
```

Файл gui.py:

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from tkinter import ttk
import tkinter as tk
import numpy as np
import backward_euler
import forward_euler
import runge_kutta
import heun
import adams_bashforth_moulton
import os

class Plotter(FigureCanvasTkAgg):

    def __init__(self, master):

        self.figure = Figure(dpi=100)
        super().__init__(self.figure, master=master)
        self.axes = self.figure.add_subplot(111)
        self.get_tk_widget().grid(column=0, row=0, sticky='nsew')

    def draw_lists(self, flag):
```

```
        self.axes.clear()

        x = [[],[],[],[]]
        i = 0

        f = open('tmp.txt', 'r')
        for line in f:
            if (line == '\n'):
                    continue
            for elem in line.split(' '):
                if (elem == '\n'):
                    continue
                x[i].append(float(elem))
            i+=1
        f.close()

        self.axes.plot(x[0], x[1], color='y')
        self.axes.plot(x[2], x[3], color='b')
        self.axes.set_xlabel('Time')
        self.axes.set_ylabel('Temperature')
        self.draw_idle()

class MainApplication(ttk.Frame):

    def __init__(self, master, *args, **kwargs):

        super().__init__(master)
        self.grid(column=0, row=0, sticky='nsew')

        frame = ttk.Frame(self, borderwidth=8)
        frame.grid(column=0, row=0, sticky='nsew')
        frame.rowconfigure(0, weight=1)
```

```
        notes = ttk.Notebook(frame)
        notes.grid(column=0, row=0, sticky='nsew')
        notes.rowconfigure(0, weight=1)

        page = ttk.Frame(notes)
        notes.add(page, text='Picture')



        self.plot = Plotter(page)

        input_frame = ttk.Frame(self)
        input_frame.grid(column=1, row=0, sticky='nsew')

        label_top_temp = ttk.Label(input_frame)
        self.slider_top_temp = ttk.Scale(input_frame, from_ = 20, to_ = 200,
                            command=lambda x:
                            label_top_temp.config(text = "Top Temp = " +
str(int(self.slider_top_temp.get())))))
        self.slider_top_temp.set(100)
        label_top_temp.config(text = "Top Temp = " +
str(int(self.slider_top_temp.get())))

        label_env_temp = ttk.Label(input_frame)
        self.slider_env_temp = ttk.Scale(input_frame, from_ = 0, to_ = 180,
                            command=lambda x:
                            label_env_temp.config(text = "Enviroment Temp = " +
str(int(self.slider_env_temp.get())))))
        self.slider_env_temp.set(50)
        label_env_temp.config(text = "Enviroment Temp = " +
str(int(self.slider_env_temp.get())))

        label_time = ttk.Label(input_frame)
        self.slider_time = ttk.Scale(input_frame, from_ = 20, to_ = 1000,
                            command=lambda x:
                            label_time.config(text = "Time = " +
str(int(self.slider_time.get())))))
```

```
        self.slider_time.set(200)
        label_time.config(text = "Time = " + str(int(self.slider_time.get())))

        self.label_err_msg = ttk.Label(input_frame)
        self.label_err_msg.config(text = "")

        button_BE = ttk.Button(input_frame, text='Backward Euler', command =
self.button_BE_clicked)

        button_FE = ttk.Button(input_frame, text='Forward Euler', command =
self.button_FE_clicked)

        button_RK = ttk.Button(input_frame, text='Runge Kutt', command =
self.button_RK_clicked)

        button_H = ttk.Button(input_frame, text='Heun', command =
self.button_H_clicked)

        button_ADM = ttk.Button(input_frame, text='Adams-Bashforth-Moulton',
command = self.button_ADM_clicked)

        button_discard_param = ttk.Button(input_frame, text='Discard params',
command = self.button_discard_param_clicked)

        button_BE.grid(column=0, row=0, columnspan=2, sticky='ew')
        button_FE.grid(column=0, row=1, columnspan=2, sticky='ew')
        button_RK.grid(column=0, row=2, columnspan=2, sticky='ew')
        button_H.grid(column=0, row=3, columnspan=2, sticky='ew')
        button_ADM.grid(column=0, row=4, columnspan=2, sticky='ew')
        label_top_temp.grid(column=0, row=5, columnspan=2, sticky='ew')
        self.slider_top_temp.grid(column=0, row=6, columnspan=2, sticky='ew')
        label_env_temp.grid(column=0, row=7, columnspan=2, sticky='ew')
        self.slider_env_temp.grid(column=0, row=8, columnspan=2, sticky='ew')
        label_time.grid(column=0, row=9, columnspan=2, sticky='ew')
        self.slider_time.grid(column=0, row=10, columnspan=2, sticky='ew')
        button_discard_param.grid(column=0, row=11, columnspan=2, sticky='ew')
```

```
        self.label_err_msg.grid(column=0, row=12, columnspan=2, sticky='ew')


    def button_BE_clicked(self):
        if (self.check_sliders()):
            self.plot.draw_lists(backward_euler.BackwardEuler(
                0.2, int(self.slider_time.get()), self.slider_top_temp.get(),
self.slider_env_temp.get()
                ).execute())


    def button_FE_clicked(self):
        if (self.check_sliders()):
            self.plot.draw_lists(forward_euler.ForwardEuler(
                0.2, int(self.slider_time.get()), self.slider_top_temp.get(),
self.slider_env_temp.get()
                ).execute())


    def button_RK_clicked(self):
        if (self.check_sliders()):
            self.plot.draw_lists(runge_kutta.Runge_Kutt(
                0.2, int(self.slider_time.get()), self.slider_top_temp.get(),
self.slider_env_temp.get()
                ).execute())


    def button_H_clicked(self):
        if (self.check_sliders()):
            self.plot.draw_lists(heun.Heun(
                0.2, int(self.slider_time.get()), self.slider_top_temp.get(),
self.slider_env_temp.get()
                ).execute())


    def button_ADM_clicked(self):
        if (self.check_sliders()):
            self.plot.draw_lists(adams_bashforth_moulton.ABM(
                0.2, int(self.slider_time.get()), self.slider_top_temp.get(),
self.slider_env_temp.get()
                ).execute())
```

```
    def button_discard_param_clicked(self):
        self.slider_env_temp.set(50)
        self.slider_top_temp.set(100)
        self.slider_time.set(200)
        self.check_sliders()

    def check_sliders(self):
        # print("hi")
        if (self.slider_top_temp.get() <= self.slider_env_temp.get()):
            self.label_err_msg.config(text = "\n\nStatus: ERROR!\nTop Temp <= Env
Temp")
            return False
        else:
            self.label_err_msg.config(text = "\n\nStatus: DONE!")
            return True

    def __del__(self):
        os.remove('tmp.txt')
        pass
```

Файл adams_bashforth_moulton.py:

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

class ABM:
    def __init__(self, _h = 0.2, _x = 200, _y0 = 100, env_temp_ = 50):
        self.h = _h
        self.x = np.array([0.0, _x])
        self.y0 = np.array([_y0])
        self.start_temp = _y0
        self.env_temp = env_temp_
        self.f = open('tmp.txt', 'w')
```

```
def __del__(self):
    self.f.close()
    pass

def RungeKutta4thOrder(self, x):
    y_len = len(self.y0)
    x_len = int((x[-1] - x[0]) / self.h)

    x = x[0]
    y = self.y0

    xsol = np.empty((0))
    xsol = np.append(xsol, x)

    y_res = np.empty((0))
    y_res = np.append(y_res, y)

    for i in range(x_len):
        k1 = mf.myFunc(y, self.env_temp)

        yp2 = y + k1*(self.h/2)

        k2 = mf.myFunc(yp2, self.env_temp)

        yp3 = y + k2*(self.h/2)

        k3 = mf.myFunc(yp3, self.env_temp)

        yp4 = y + k3*self.h

        k4 = mf.myFunc(yp4, self.env_temp)

        for j in range(y_len):
            y[j] = y[j] + (self.h/6)*(k1[j] + 2*k2[j] + 2*k3[j] + k4[j])
```

```
            x = x + self.h
            xsol = np.append(xsol, x)

            for r in range(len(y)):
                y_res = np.append(y_res, y[r])

        return [xsol, y_res]


    def ABM4thOrder(self):

        y_len = len(self.y0)

        dx = int((self.x[-1] - self.x[0]) / self.h)

        xrk = [self.x[0] + k * self.h for k in range(dx + 1)]

        [xx, yy] = self.RungeKutta4thOrder((xrk[0], xrk[3]))

        self.x = xx
        xsol = np.empty(0)
        xsol = np.append(xsol, self.x)

        y = yy
        yn = np.array([yy[0]])
        y_res = np.empty(0)
        y_res = np.append(y_res, y)

        for i in range(3, dx):
            x00 = self.x[i]; x11 = self.x[i-1]; x22 = self.x[i-2]; x33 =
self.x[i-3]; xpp = self.x[i]+self.h

            y00 = np.array([y[i]])
            y11 = np.array([y[i - 1]])
```

```
            y22 = np.array([y[i - 2]])
            y33 = np.array([y[i - 3]])

            y0prime = mf.myFunc(y00, self.env_temp)
            y1prime = mf.myFunc(y11, self.env_temp)
            y2prime = mf.myFunc(y22, self.env_temp)
            y3prime = mf.myFunc(y33, self.env_temp)

            ypredictor = y00 + (self.h/24)*(55*y0prime - 59*y1prime + 37*y2prime
- 9*y3prime)
            ypp = mf.myFunc(ypredictor, self.env_temp)

            for j in range(y_len):
                yn[j] = y00[j] + (self.h/24)*(9*ypp[j] + 19*y0prime[j] -
5*y1prime[j] + y2prime[j])

            xs = self.x[i] + self.h
            xsol = np.append(xsol, xs)

            self.x = xsol

            for r in range(len(yn)):
                y_res = np.append(y_res, yn)

            y = y_res

        return [xsol, y_res]

    def execute(self):
        [ts, ys] = self.ABM4thOrder()
        for index in ts:
            self.f.write(str(index) + ' ')
        self.f.write('\n')
        for index in ys:
            self.f.write(str(index) + ' ')
```

```
        self.f.write('\n')


        t = np.arange(0, self.x[-1], 1)
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')


        for i in t:
            y_math_res = self.env_temp + (self.start_temp - self.env_temp) *
np.e**(-(np.log(2) * i/25))
            self.f.write(str(y_math_res) + ' ')
        self.f.write('\n')


        return 0
```

Файл backward_euler.py:

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


class BackwardEuler:
    def __init__(self, _h = 0.2, _x = 200, _y0 = 100, env_temp_ = 50):
        self.h = _h
        self.x = np.array([0.0, _x])
        self.y0 = np.array([_y0])
        self.start_temp = _y0
        self.env_temp = env_temp_
        self.f = open('tmp.txt', 'w')


    def __del__(self):
        self.f.close()
        pass


    def main_BE(self):
        y_len = len(self.y0)
```

```
    x_len = int((self.x[-1] - self.x[0])/self.h)

    x = self.x[0]
    y = self.y0

    self.f.write(str(x) + ' ')

    y_res = np.empty(0)
    y_res = np.append(y_res, y)

    for i in range(x_len):
        F_x_t = mf.myFunc(y, self.env_temp)/(1+self.h)
        # print(">> ", x, h, y, F_x_t)

        for j in range(y_len):
            y[j] = y[j] + self.h*F_x_t[j]

        x += self.h
        self.f.write(str(x) + ' ')

        for r in range(len(y)):
            y_res = np.append(y_res, y[r])

    self.f.write('\n')

    return y_res

def execute(self):
    ys = self.main_BE()

    for index in ys:
        self.f.write(str(index) + ' ')
    self.f.write('\n')
```

```
        t = np.arange(0, self.x[-1], 1)
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        for i in t:
            y_math_res = self.env_temp + (self.start_temp - self.env_temp) *
np.e**(-(np.log(2) * i/25))
            self.f.write(str(y_math_res) + ' ')
        self.f.write('\n')

        return 0
```

Файл forward_euler.py:

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

class ForwardEuler:
    def __init__(self, _h = 0.2, _x = 200, _y0 = 100, env_temp_ = 50):
        self.h = _h
        self.x = np.array([0.0, _x])
        self.y0 = np.array([_y0])
        self.start_temp = _y0
        self.env_temp = env_temp_
        self.f = open('tmp.txt', 'w')

    def __del__(self):
        self.f.close()
        pass

    def main_FE(self):
        y_len = len(self.y0)
        x_len = int((self.x[-1] - self.x[0])/self.h)
```

```
        x = self.x[0]
        y = self.y0

        self.f.write(str(x) + ' ')

        y_res = np.empty(0)
        y_res = np.append(y_res, y)

        for i in range(x_len):
            F_x_t = mf.myFunc(y, self.env_temp)

            for j in range(y_len):
                y[j] = y[j] + self.h*F_x_t[j]

            x += self.h
            self.f.write(str(x) + ' ')

            for r in range(len(y)):
                y_res = np.append(y_res, y[r])

        self.f.write('\n')

        return y_res

    def execute(self):
        ys = self.main_FE()
        for index in ys:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        t = np.arange(0, self.x[-1], 1)
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')
```

```
        for i in t:
            y_math_res = self.env_temp + (self.start_temp - self.env_temp) *
np.e**(-(np.log(2) * i/25))
            self.f.write(str(y_math_res) + ' ')
        self.f.write('\n')


        return 0
```

Файл heun.py:

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



class Heun:
    def __init__(self, _h = 0.2, _x = 200, _y0 = 100, env_temp_ = 50):
        self.h = _h
        self.x = np.array([0.0, _x])
        self.y0 = np.array([_y0])
        self.start_temp = _y0
        self.env_temp = env_temp_
        self.f = open('tmp.txt', 'w')

    def __del__(self):
        self.f.close()
        pass

    def main_H(self, y0, x_range, h):
        y_len = len(y0)
        x_len = int((x_range[-1] - x_range[0])/h)

        x = x_range[0]
        y = y0
```

```
        self.f.write(str(x) + ' ')

        y_res = np.empty(0)
        y_res = np.append(y_res, y)

        for i in range(x_len):
            y0prime = mf.myFunc(y, self.env_temp)

            k1 = y0prime * h

            ypredictor = y + k1

            y1prime = mf.myFunc(ypredictor, self.env_temp)

            for j in range(y_len):
                y[j] = y[j] + (h/2)*y0prime[j] + (h/2)*y1prime[j]

            x = x + h
            self.f.write(str(x) + ' ')

            for r in range(len(y)):
                y_res = np.append(y_res, y[r])

        self.f.write('\n')

        return y_res

    def execute(self):
        ys = self.main_H(self.y0, self.x, self.h)
        for index in ys:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        t = np.arange(0, self.x[-1], 1)
```

```
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        for i in t:
            y_math_res = self.env_temp + (self.start_temp - self.env_temp) *
np.e**(-(np.log(2) * i/25))
            self.f.write(str(y_math_res) + ' ')
        self.f.write('\n')

        return 0
```

Файл runge_kutta.py:

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

class Runge_Kutt:
    def __init__(self, _h = 0.2, _x = 200, _y0 = 100, env_temp_ = 50):
        self.h = _h
        self.x = np.array([0.0, _x])
        self.y0 = np.array([_y0])
        self.start_temp = _y0
        self.env_temp = env_temp_
        self.f = open('tmp.txt', 'w')

    def __del__(self):
        self.f.close()
        pass

    def RKF45(self, y0, x_range, h):
        y_len = len(y0)
        x_len = int((x_range[-1] - x_range[0])/h)
```

```
        x = x_range[0]
        y = y0

        self.f.write(str(x) + ' ')

        y_res = np.empty(0)
        y_res = np.append(y_res, y)

        for i in range(x_len):
            k1 = mf.myFunc(y, self.env_temp)

            yp2 = y + k1*(h/5)

            k2 = mf.myFunc(yp2, self.env_temp)

            yp3 = y + k1*(3*h/40) + k2*(9*h/40)

            k3 = mf.myFunc(yp3, self.env_temp)

            yp4 = y + k1*(3*h/10) - k2*(9*h/10) + k3*(6*h/5)

            k4 = mf.myFunc(yp4, self.env_temp)

            yp5 = y - k1*(11*h/54) + k2*(5*h/2) - k3*(70*h/27) + k4*(35*h/27)

            k5 = mf.myFunc(yp5, self.env_temp)

            yp6 = y + k1*(1631*h/55296) + k2*(175*h/512) + k3*(575*h/13824) +
k4*(44275*h/110592) + k5*(253*h/4096)

            k6 = mf.myFunc(yp6, self.env_temp)

            for j in range(y_len):
                y[j] = y[j] + h*(37*k1[j]/378 + 250*k3[j]/621 + 125*k4[j]/594 +
512*k6[j]/1771)
```

```
            x = x + h
            self.f.write(str(x) + ' ')

            for r in range(len(y)):
                y_res = np.append(y_res, y[r])

        self.f.write('\n')

        return y_res

    def execute(self):
        ys = self.RKF45(self.y0, self.x, self.h)
        for index in ys:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        t = np.arange(0, self.x[-1], 1)
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')

        for i in t:
            y_math_res = self.env_temp + (self.start_temp - self.env_temp) *
np.e**(-(np.log(2) * i/25))
            self.f.write(str(y_math_res) + ' ')
        self.f.write('\n')

        return 0
```

Файл my_func.py:

```
import numpy as np

def myFunc(y, env_temp):
    dy = np.zeros((len(y)))
```

```
        # dy[0] = 3*(1+x) - y[0]
        dy[0] = 1/25 * np.log(2) * (env_temp - y)
    return dy
```