**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.  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**

[**ЗАДАНИЕ** 2](#_Toc74612826)

[**НА КУРСОВУЮ РАБОТУ** 2](#_Toc74612827)

[**АННОТАЦИЯ** 3](#_Toc74612828)

[**Введение** 5](#_Toc74612829)

[**Закон Ньютона-Рихмана** 6](#_Toc74612830)

[**Прямой метод Эйлера** 8](#_Toc74612831)

[**Обратный метод Эйлера** 9](#_Toc74612832)

[**Метод Рунге-Кутты-Фельберга 4-5 порядка** 10](#_Toc74612833)

[**Метод Хойна** 11](#_Toc74612834)

[**Метод Адамса-Башфорта** 12](#_Toc74612835)

[**GUI** 14](#_Toc74612836)

[**Вывод** 18](#_Toc74612837)

[**Используемая литература** 19](#_Toc74612838)

[**ПРИЛОЖЕНИЕ А. Код программы** 20](#_Toc74612839)

**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:

, при k < 0

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:

Умножим уравнение на и получим:

Проинтегрируем обе части уравнения:

Получаем:

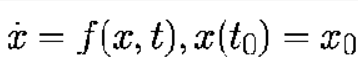
Избавимся от логарифма и получим:

Раскроем модуль:

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

**Прямой метод Эйлера**

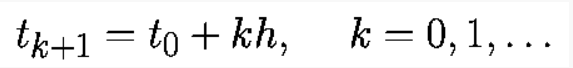
Пусть дана система дифференциальных уравнений с непрерывным временем:

,

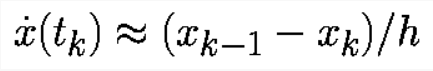
последовательностью точек x0,x1,... в соответствующие моменты времени t0,t1,... Значения точек должны удоволетворять приближенному равенству:



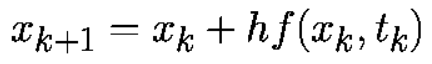
Если специально не оговорено иное, то предполагается, что моменты времени выбираются через равные интервалы с величиной шага h>0, то есть



Аппроксимируем производную в момент времени tk соотношением

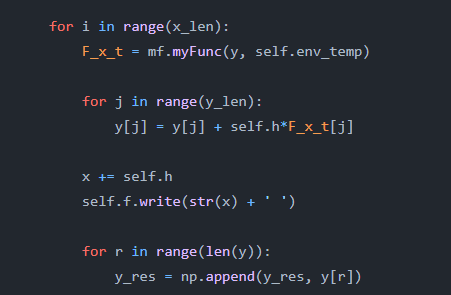


При такой аппроксимации уравнение примет вид:

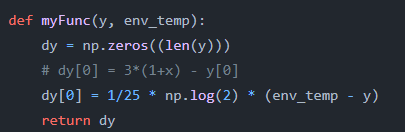
,

Эта формула известна как прямой метод Эйлера.

В нашем коде этот метод был реализован так:



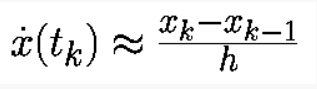
F\_x\_t – функция f(x, t), self.h – это длина шага по x, y[j] – температура на текущем шаге, my\_func.py выглядит так:

,

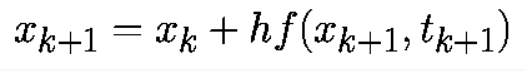
Где env\_temp – это температура окружения, а 1/25 \* np.log(2) – k.

**Обратный метод Эйлера**

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

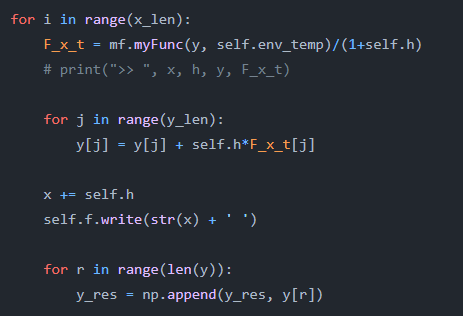


Такая аппроксимация дает формулу обратного метода Эйлера:



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

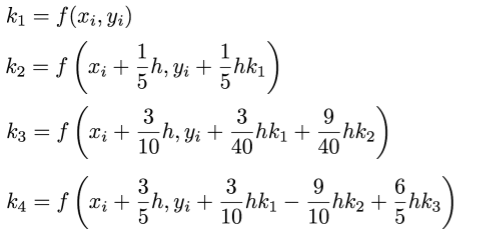
В нашем коде этот метод был реализован так:

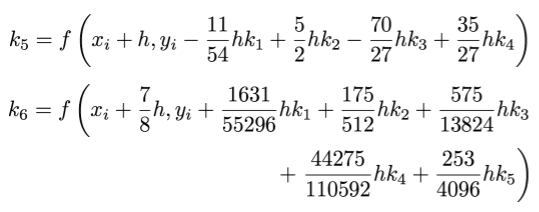


Где F\_x\_t – функция f(x, t), self.h – это длина шага по x, y[j] – температура на текущем шаге, self.env\_temp – температура окружения.

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

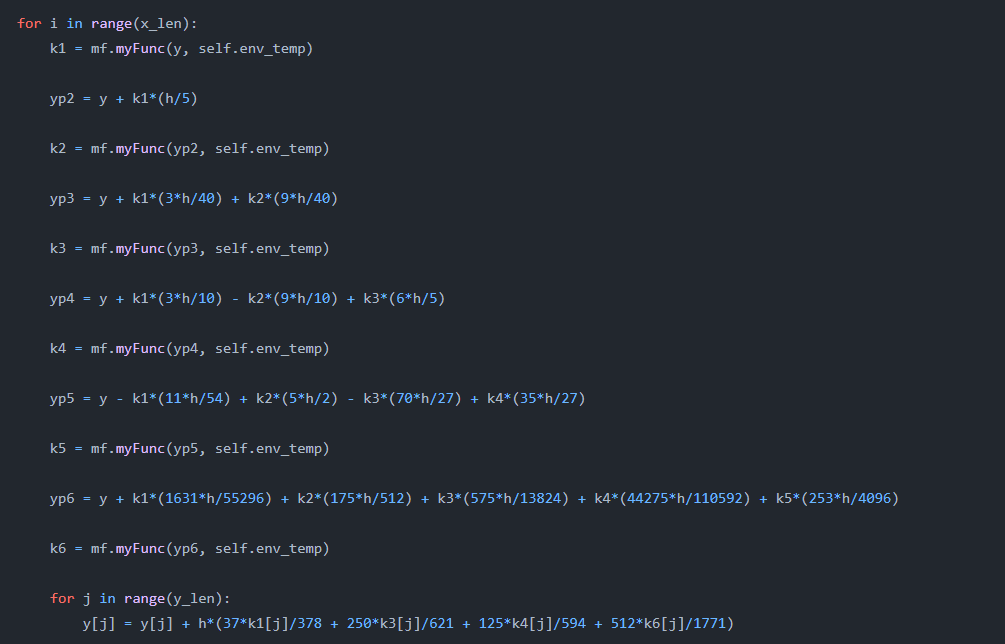
Метод Рунге-Кутты-Фельберга 4-5 порядка – адаптивный метод, в котором каждый шаг требует использования шести следующих значений:





Аппроксимация для этого метода выглядит так:  


Этот метод был реализован таким образом:

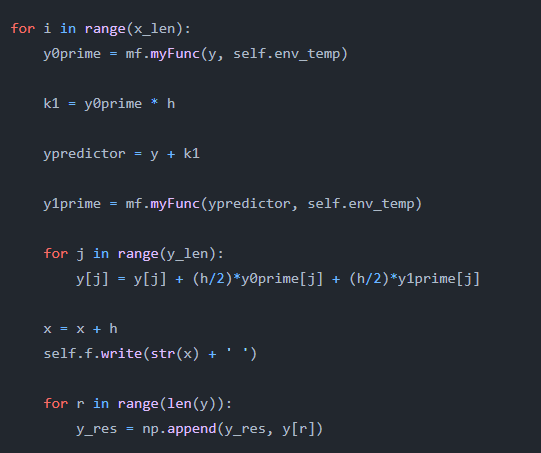


**Метод Хойна**

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:



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.

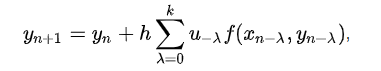
Пусть дана система дифференциальных уравнений первого порядка



для которой надо найти решение на сетке с постоянным шагом



Формула метода Адамса для решения этой системы имеет вид:



Явные методы Адамса — Башфорта:

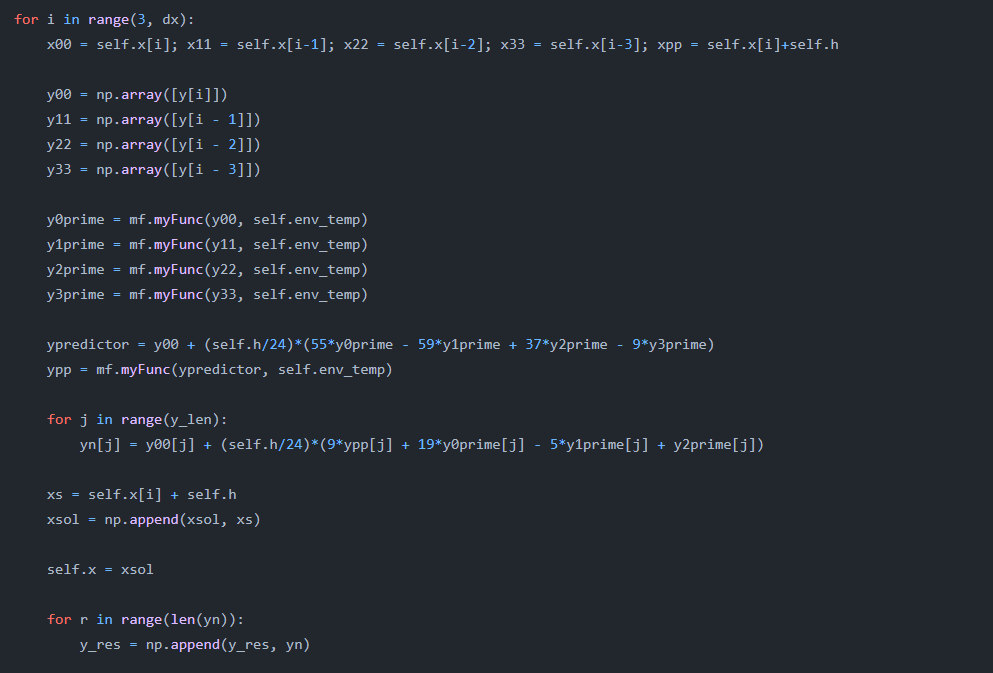








Для предварительного вычисления решения в k начальных точках был использован метод Рунге-Кутты 4-ого порядка. Сам же метод Адамса-Башфорта в работе выглядит так:



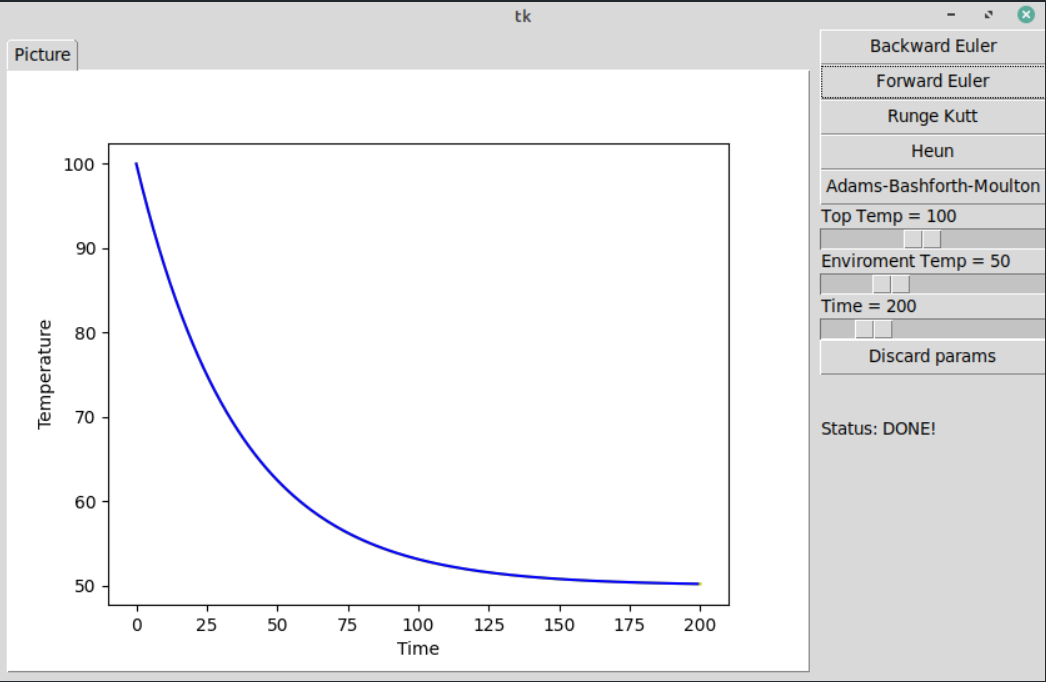
**GUI**

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

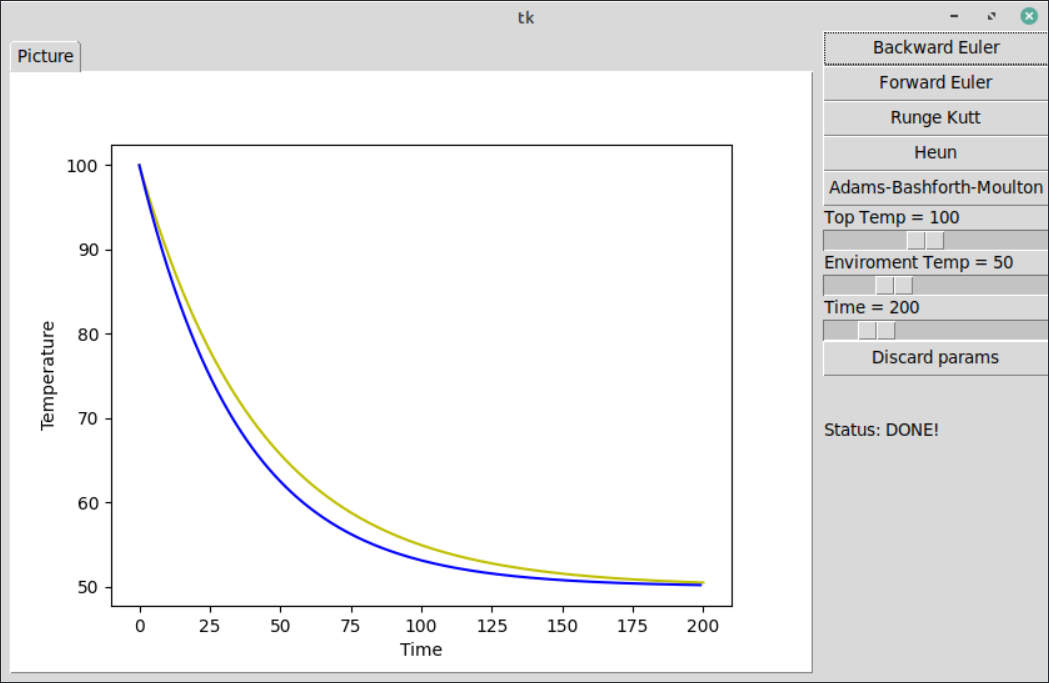
Интерфейс программы включает в себя 6 кнопок, 3 слайдера, окно для графиков и лэйбл с выводом информации об успешном/неуспешном запуске программы:

Первые пять кнопок в интерфейсе вызывают численные методы:

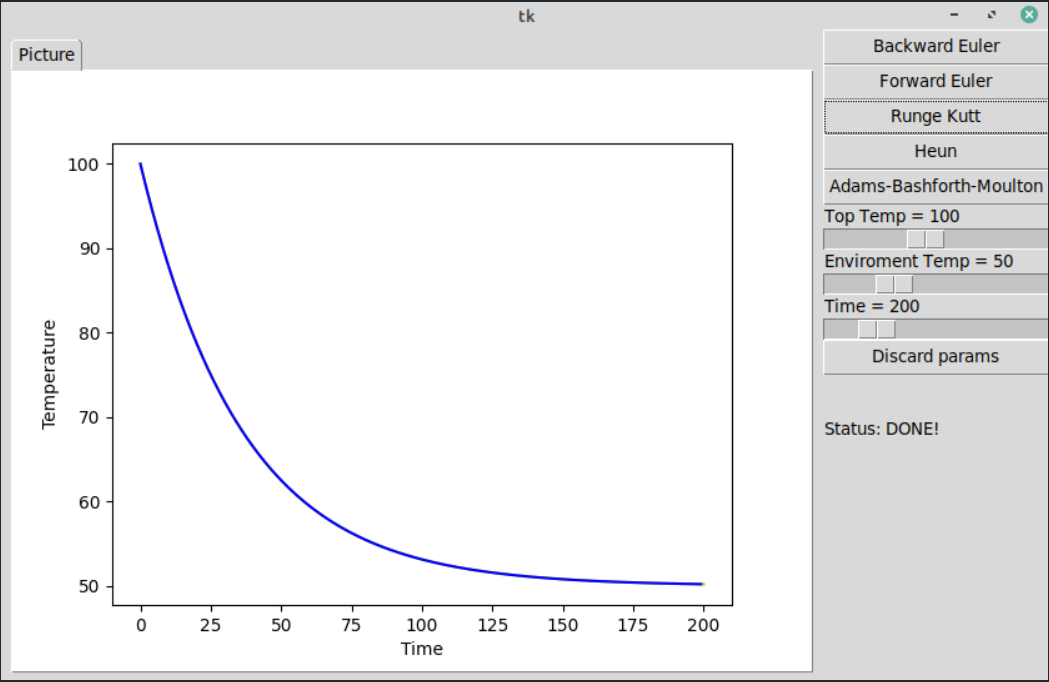
Forward Euler:



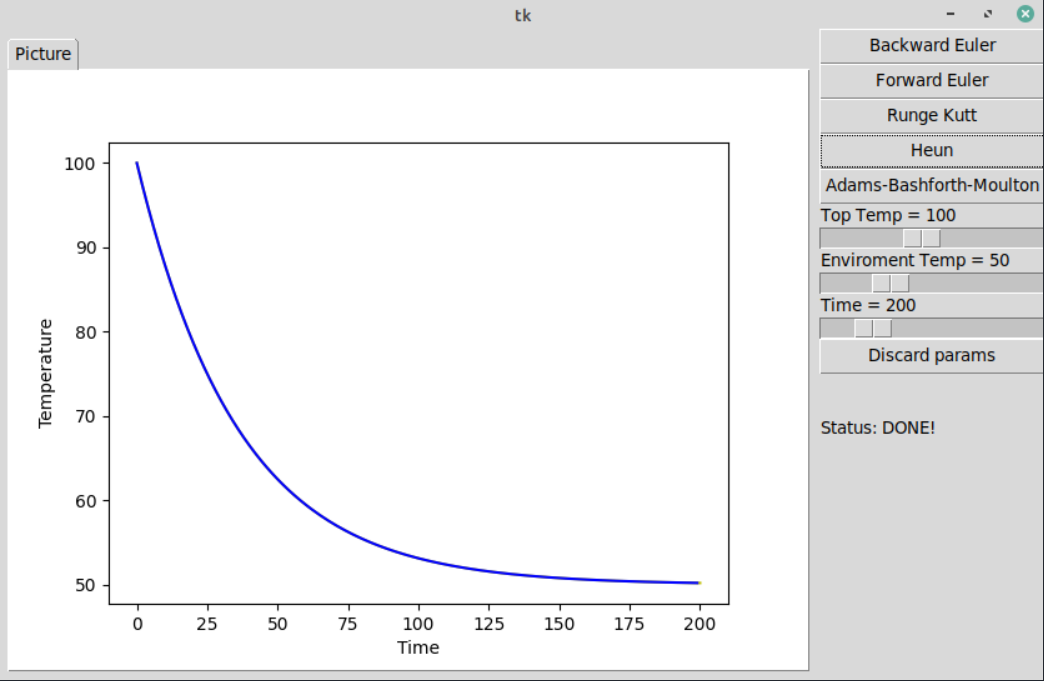
Backward Euler:



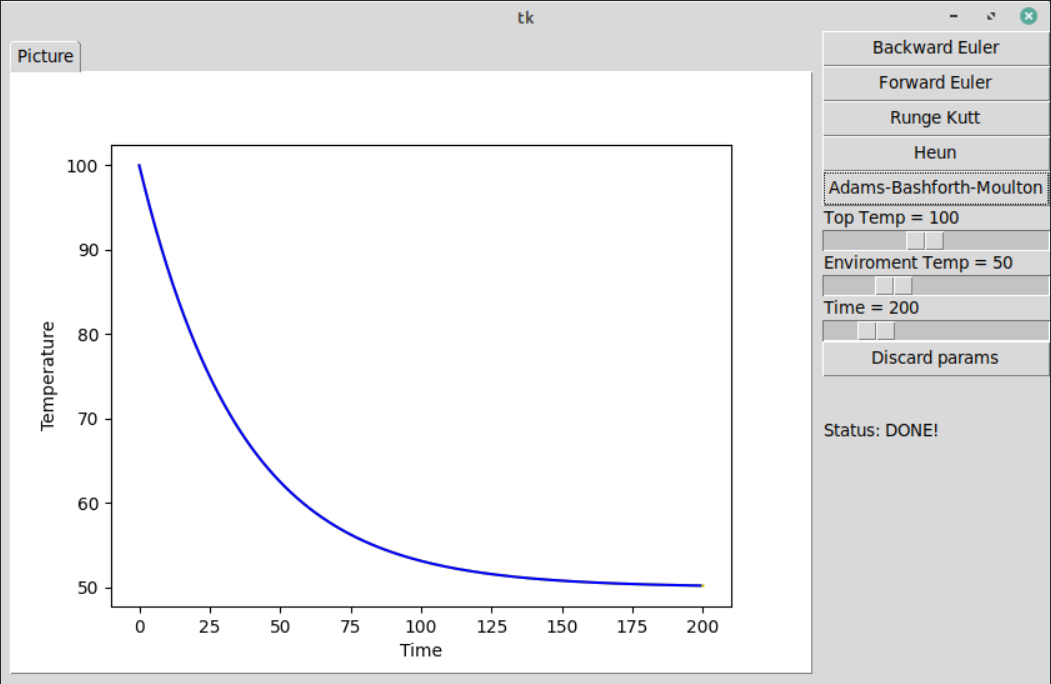
Runge Kutt:



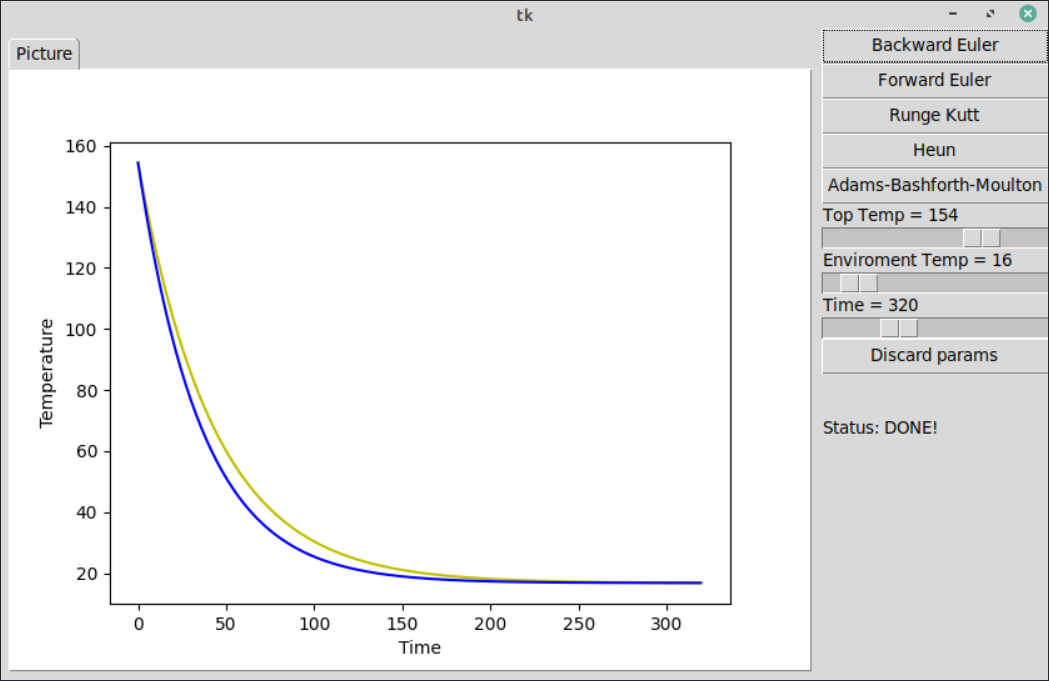
Heun:



Adams-Bashfourth:

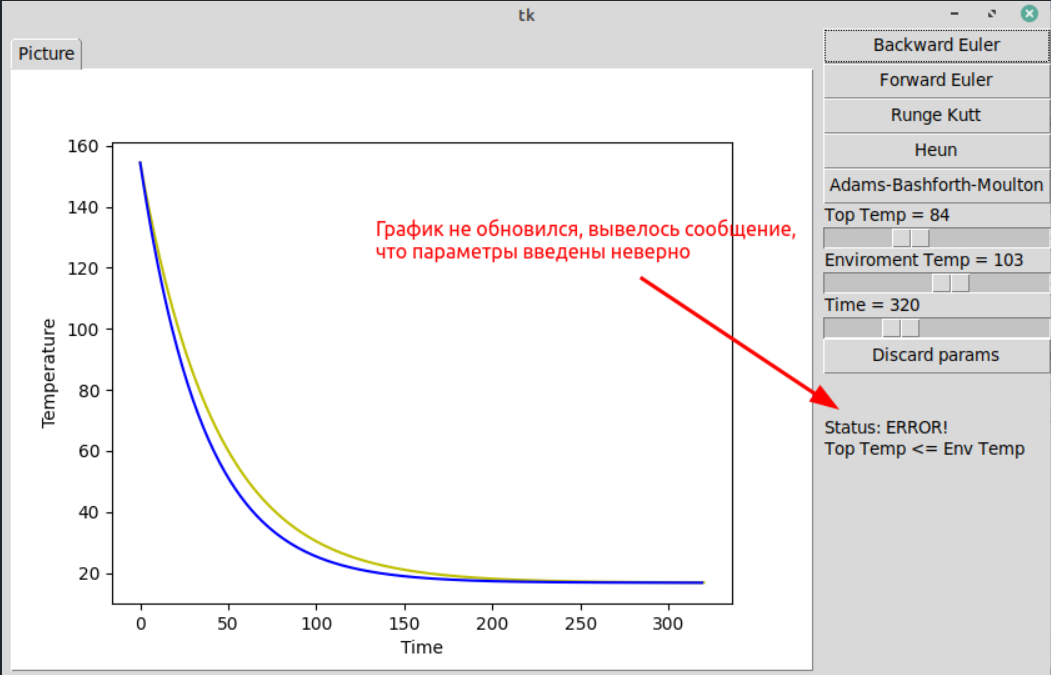


С помощью слайдеров можно менять условия поставленной задачи:



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