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

КУРСОВАЯ РАБОТА

по дисциплине «Дифференциальные уравнения»
Тема: Соскальзывание цепочки

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: Соскальзывание цепочки

Исходные данные:
Соскальзывание цепочки

Содержание пояснительной записки:
«Содержание», «Введение», «2-ой Закон Ньютона», «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 соскальзывание цепочки. Для этого использовался 2-ой Закон Ньютона, 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 chain slip. For this, the 2nd Newton 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. Соскальзывание цепочки является этим процессом. Были использованы методы интегрирования дифференциальных уравнений динамических систем для решения 2-ого закона Ньютона, такие как: «Прямой метод Ньютона», «Обратный метод Ньютона», «Метод Рунге-Кутты-Фельберга 4-5-го порядка», «Метод Хойна», «Метод Адамса».

2-ой Закон Ньютона

2-ой Закон Ньютона устанавливает связь между силой **F**, действующей на тело массы **m**, и ускорением **a**, которое приобретает тело под действием этой силы. Дифференциальная формулировка выглядит так:

$$F = \frac{dp}{dt}$$

Где **F** – сила, **t** – время, **p** – импульс.

В предположении, что движение одномерное, второй закон Ньютона в этом случае записывается в виде дифференциального уравнения второго порядка:

$$F(t) = m\frac{d^2x}{dt^2}$$

Согласно второму закону Ньютона, дифференциальное уравнение движения цепочки имеет вид:

$$m\frac{d^2x}{dt^2} = P - F\text{тр}$$

Отсюда:

$$m\frac{d^2x}{dt^2} = mg\frac{x}{L} - \mu mg\frac{L-x}{L}$$

Поделим обе части уравнения на m:

$$\frac{d^2x}{dt^2} = g\frac{x}{L} - \mu g\frac{L-x}{L}$$

Получаем:

$$\frac{d^2x}{dt^2} - x\frac{g(1+\mu)}{L} = -\mu g$$

Проинтегрируем и получим функцию f(x,t), использующуюся в численных методах.

$$\frac{g\mu x^2}{2L} + \frac{gx^2}{2L} - g\mu x$$

Где μ - коэффициент трения, L – длина цепочки, g – ускорение свободного падения.

Для получения аналитической формулы должно быть известно начальное условие:

$$x(t=0) = \frac{\mu L}{1+\mu} + \varepsilon$$

$$v(t=0) = 0$$

Длина свисающей части цепочки при равновесии составляет:

$$x = \frac{\mu L}{1+\mu}$$

Скольжение цепочки описывается законом:

$$x(t) = \frac{\varepsilon}{2} e^{\mathrm{sqrt}\left(\frac{(1+\mu)g}{L}\right)t} + \frac{\varepsilon}{2} e^{-\mathrm{sqrt}\left(\frac{(1+\mu)g}{L}\right)t} + \frac{\mu L}{1+\mu}$$

Прямой метод Эйлера

В нашем коде этот метод был реализован так:

```
def main_FE(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    x = 0
    y = {}
    y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        F_x_t = mf.myFunc(x, self.coef, self.chain_len)

        j += 1
        y[j] = y[j-1] + self.h*F_x_t
        print (y[j])

        x += self.h
        if (y[j] > 1):
            y[j] = 1
        self.f.write(str(x) + ' ')
    self.f.write('\n')

    return y
```

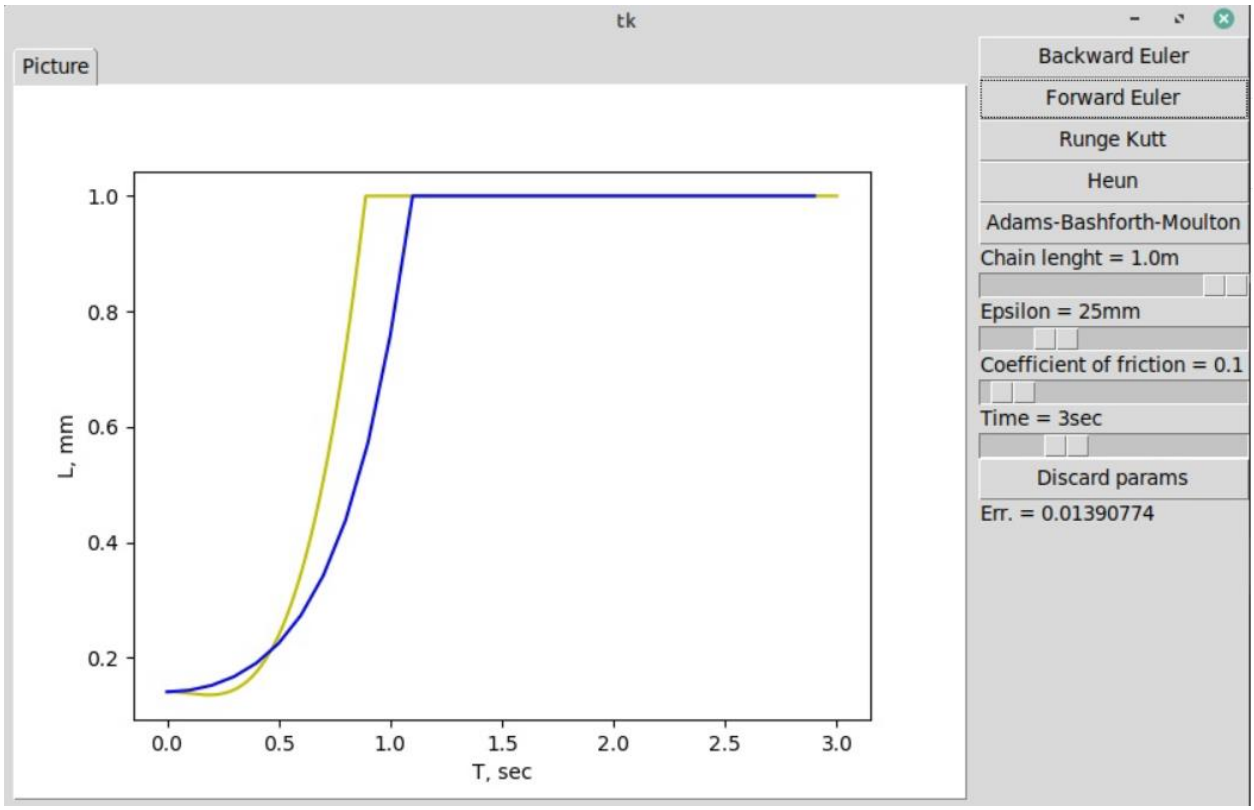
F_x_t – функция f(x, t), self.h – это длина шага по x, my_func.py выглядит так:

```
import numpy as np

def myFunc(x, mu, l):
    dy = 4.9 * x**2 * (mu + 1) / l - 9.8 * mu * x
    # dy = 5.self.time9 * (x)**2 - 0.98 * (x)
    return dy
```

,

Где mu – это коэффициент трения, а 9.8 – ускорение свободного падения.

Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



Реализация аналитического решения выглядит так (для всех методов она одинаковая):

```
for i in t:
    T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 / self.chain_len) * i) + 2 * self.eps/2000 * n
    if (T > 1):
        T = 1
    self.f.write(str(T) + ' ')
self.f.write('\n')
```

Также после выполнения метода, на экран выводится его глобальная ошибка для шага h = 0.01.

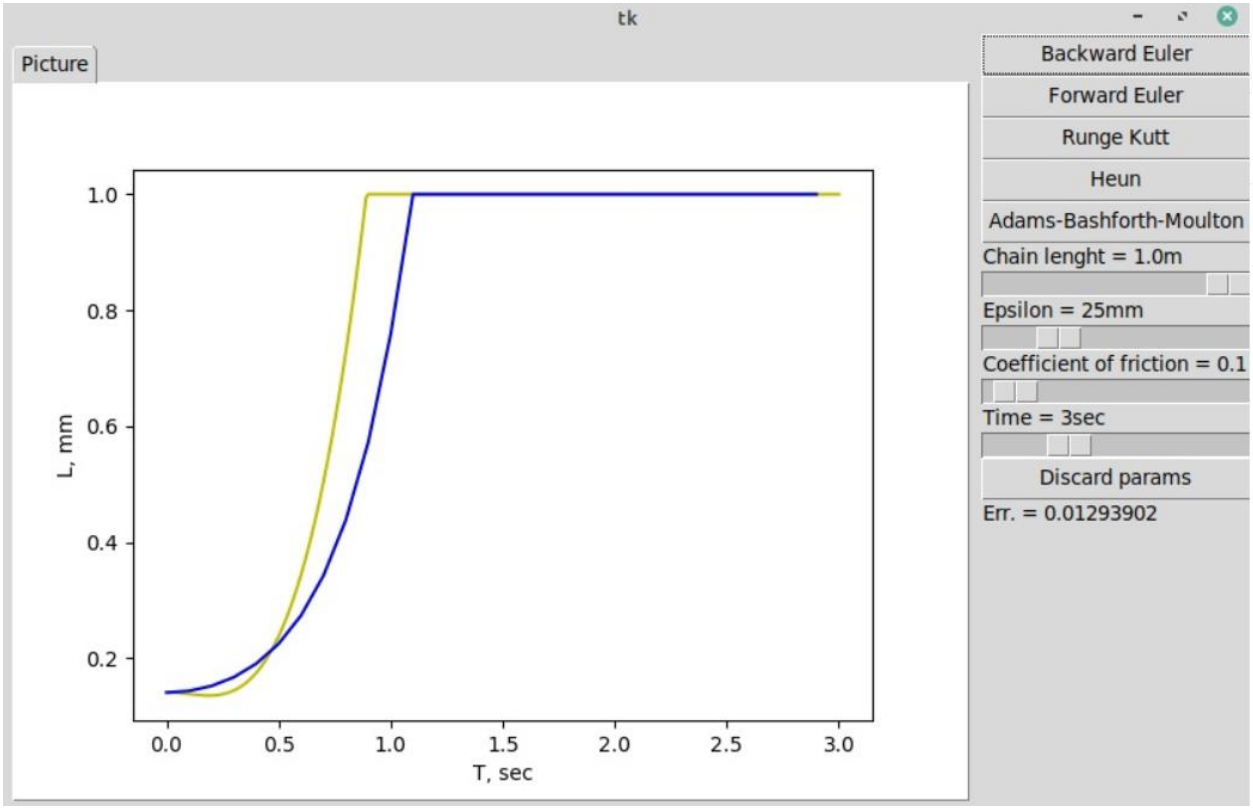
Обратный метод Эйлера

В нашем коде этот метод был реализован так:

```
def main_BE(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    x = 0
    y = {}
    y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        F_x_t = mf.myFunc(x, self.coef, self.chain_len)/(1+self.h)

        j += 1
        y[j] = y[j-1] + self.h*F_x_t
        print(y[j])

        x += self.h
        if (y[j] > 1):
            y[j] = 1
        self.f.write(str(x) + ' ')
    self.f.write('\n')

    return y
```

Где F_x_t – функция f(x, t), self.h – это длина шага по x.

Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



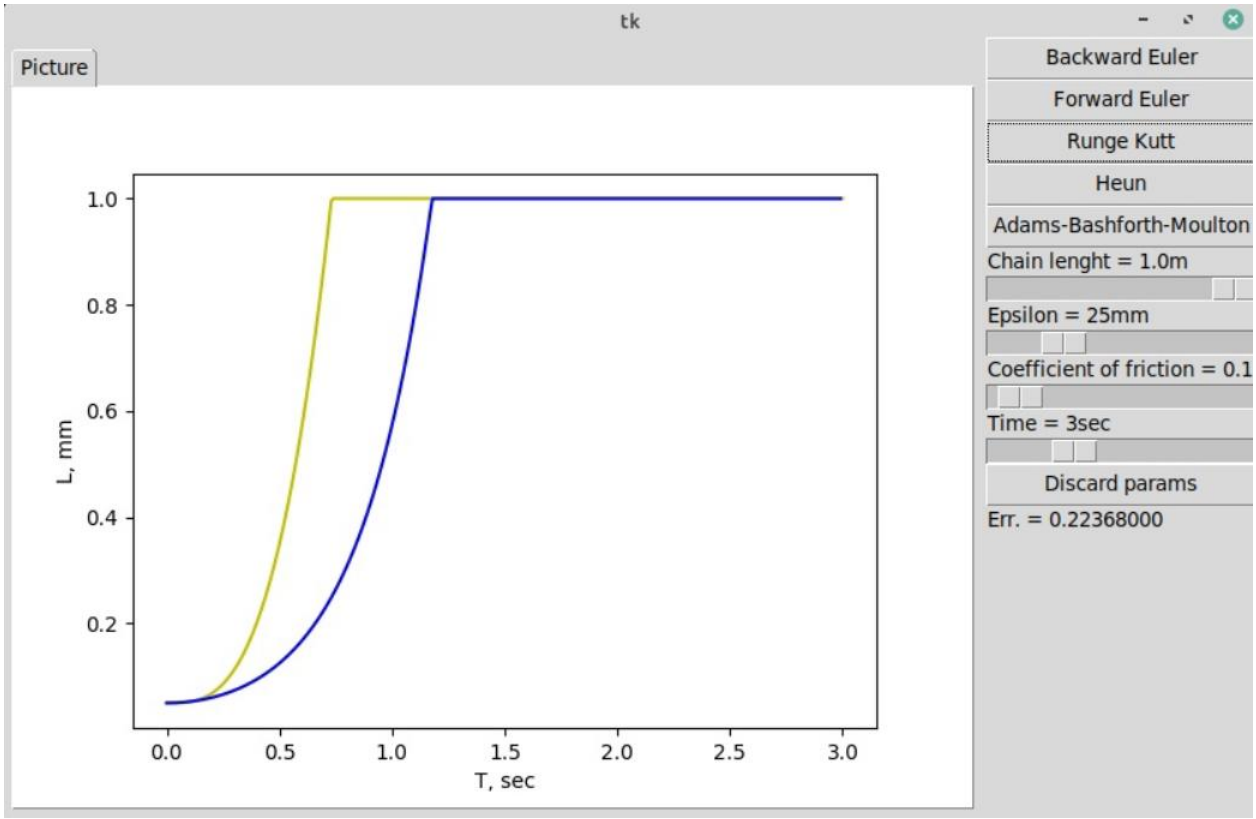
Также после выполнения метода, на экран выводится его глобальная ошибка для шага h = 0.01.

Метод Рунге-Кутты-Фельберга 4-5 порядка

Этот метод был реализован таким образом:

```
def RKF45(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    x = 0
    y = {}
    y[j] = self.coef * 1 / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        yp2 = x + mf.myFunc(x, self.coef, self.chain_len)*(self.h/5)
        yp3 = x + mf.myFunc(x, self.coef, self.chain_len)*(3*self.h/40) + mf.myFunc(yp2, self.coef, self.chain_len)*(9*self.h/40)
        yp4 = x + mf.myFunc(x, self.coef, self.chain_len)*(3*self.h/10) - mf.myFunc(yp2, self.coef, self.chain_len)*(9*self.h/10) + mf.myFunc(yp3, self.coef, self.chain_le
        yp5 = x - mf.myFunc(x, self.coef, self.chain_len)*(11*self.h/54) + mf.myFunc(yp2, self.coef, self.chain_len)*(5*self.h/2) - mf.myFunc(yp3, self.coef, self.chain_le
        yp6 = x + mf.myFunc(x, self.coef, self.chain_len)*(1631*self.h/55296) + mf.myFunc(yp2, self.coef, self.chain_len)*(175*self.h/512) + mf.myFunc(yp3, self.coef, self

        j += 1
        y[j] = y[j-1] + self.h*(37*mf.myFunc(x, self.coef, self.chain_len)/378 + 22 * self.eps*mf.myFunc(yp3, self.coef, self.chain_len)/621 + 125*mf.myFunc(yp4, self.coef

        x += self.h
        if (y[j] > 1):
            y[j] = 1
        self.f.write(str(x) + ' ')
    self.f.write('\n')

    return y
```

Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



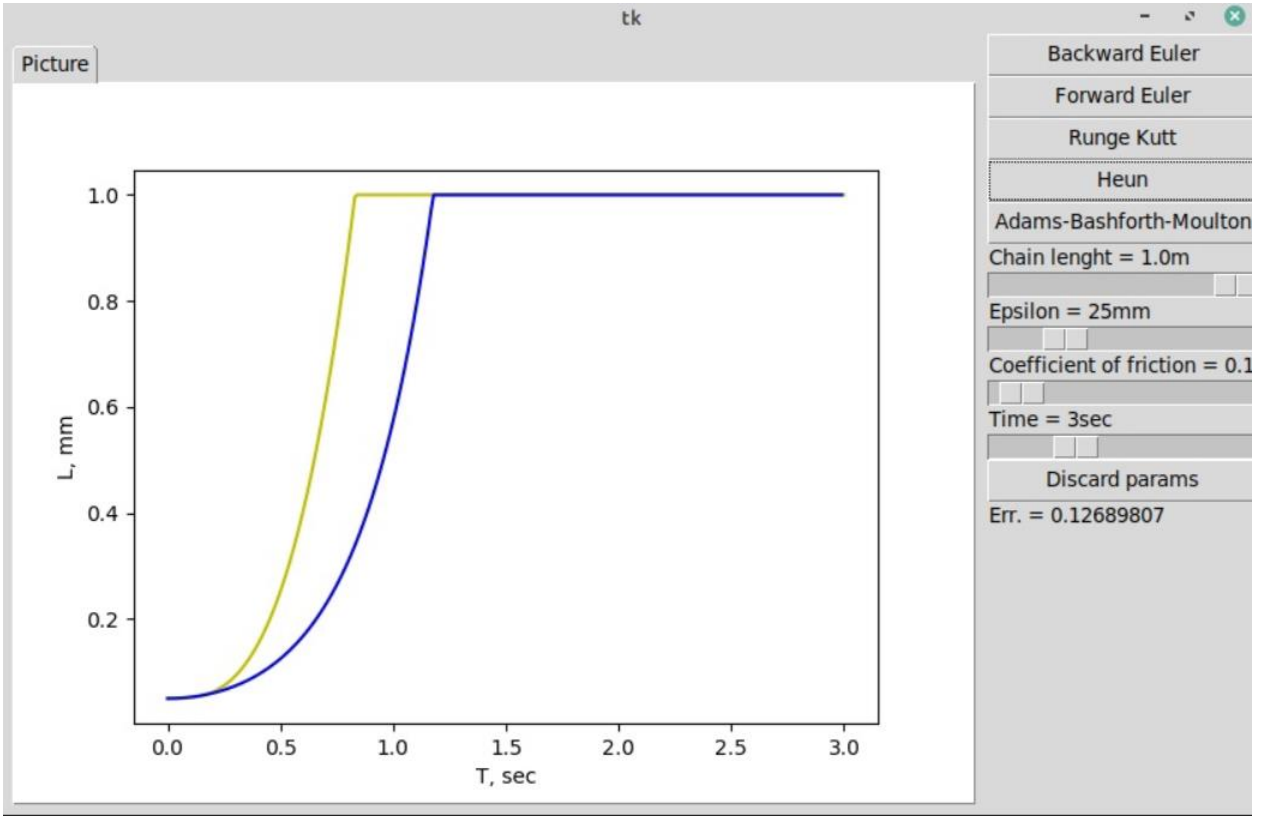
Также после выполнения метода, на экран выводится его глобальная ошибка для шага h = 0.01.

Метод Хойна

В работе метод был реализован так:

```
def main_H(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    x = 0
    y = {}
    y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        j += 1
        y[j] = y[j-1] + (self.h/2)*mf.myFunc(x, self.coef, self.chain_len) + (self.h/2)*mf.myFunc(x + mf.myFunc(x, self.coef, self.chain_len) * self.h, self.coef, self.cha
        print(y[j])
        if (y[j] > 1):
            y[j] = 1

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

    return y
```

Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



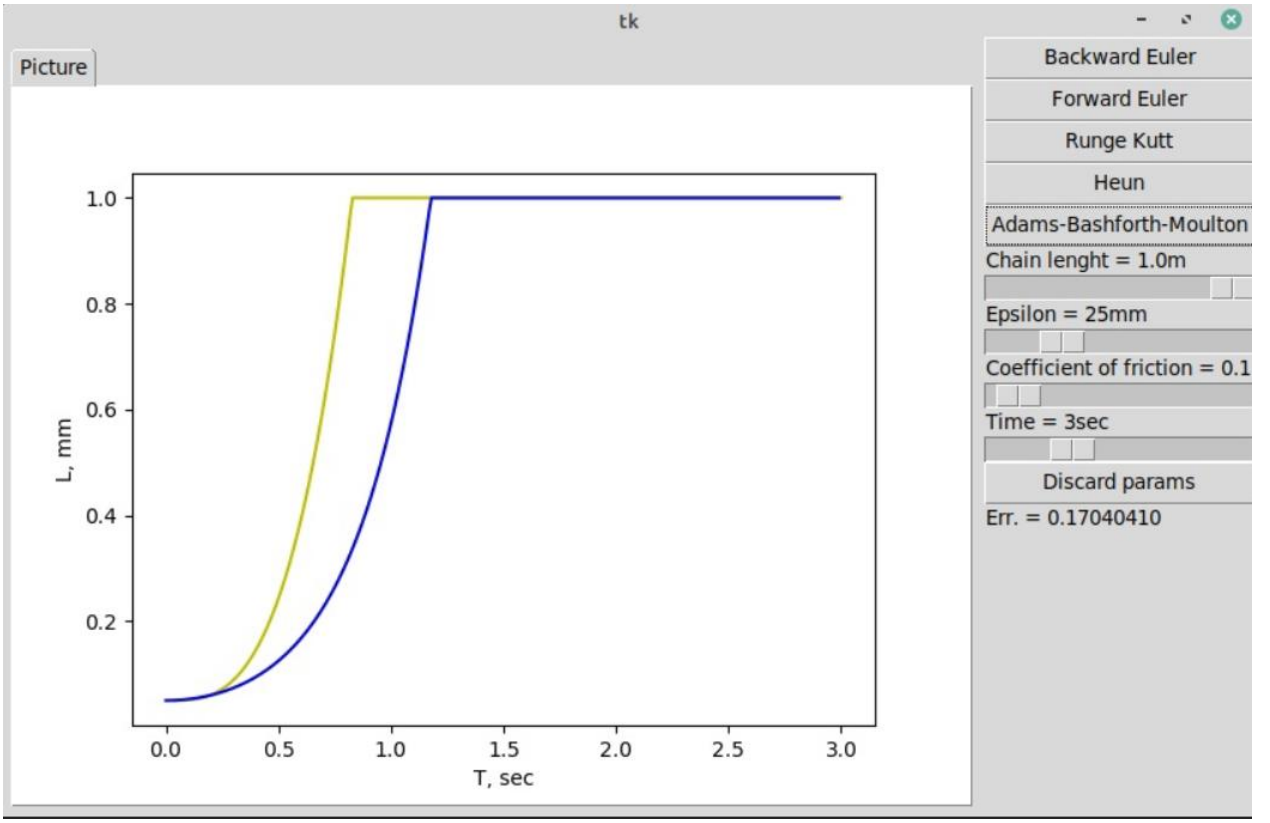
Также после выполнения метода, на экран выводится его глобальная ошибка для шага h = 0.01.

Метод Адамса-Башфорта

В работе метод был реализован так:

```
        for i in range(3, dx):
            y0prime = mf.myFunc(y[i], self.coef, self.chain_len)
            y1prime = mf.myFunc(y[i - 1], self.coef, self.chain_len)
            y2prime = mf.myFunc(y[i - 2], self.coef, self.chain_len)
            y3prime = mf.myFunc(y[i - 3], self.coef, self.chain_len)

            ypredictor = y[i] + (self.h/24)*(55*y0prime - 59*y1prime + 37*y2prime - 9*y3prime)
            ypp = mf.myFunc(ypredictor, self.coef, self.chain_len)

            yn = y[i] + (self.h/24)*(9*ypp + 19*y0prime - 5*y1prime + y2prime)

            if (yn > 1):
                yn = 1
            # print (yn)

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

            self.x = xsol

            y_res = np.append(y_res, yn)
        return [xsol, y_res]
```

Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



Также после выполнения метода, на экран выводится его глобальная ошибка для шага h = 0.01.

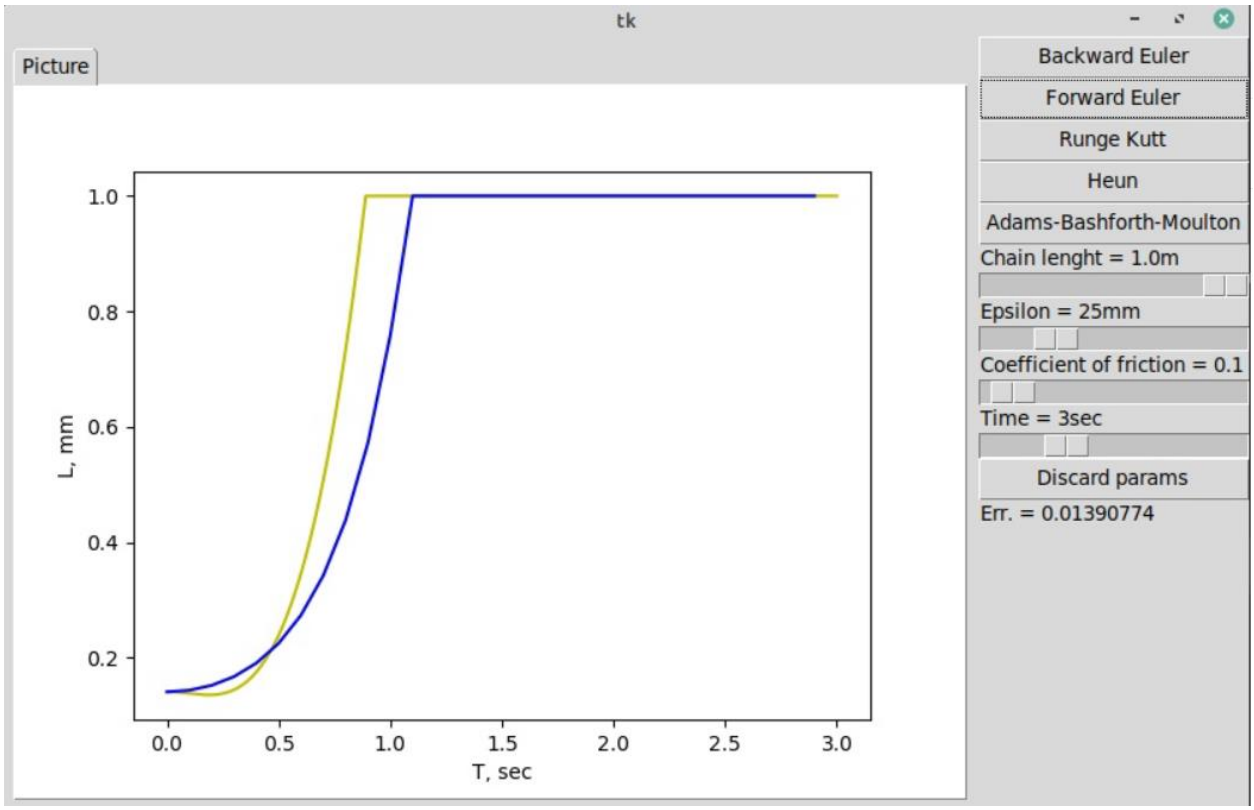
GUI

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

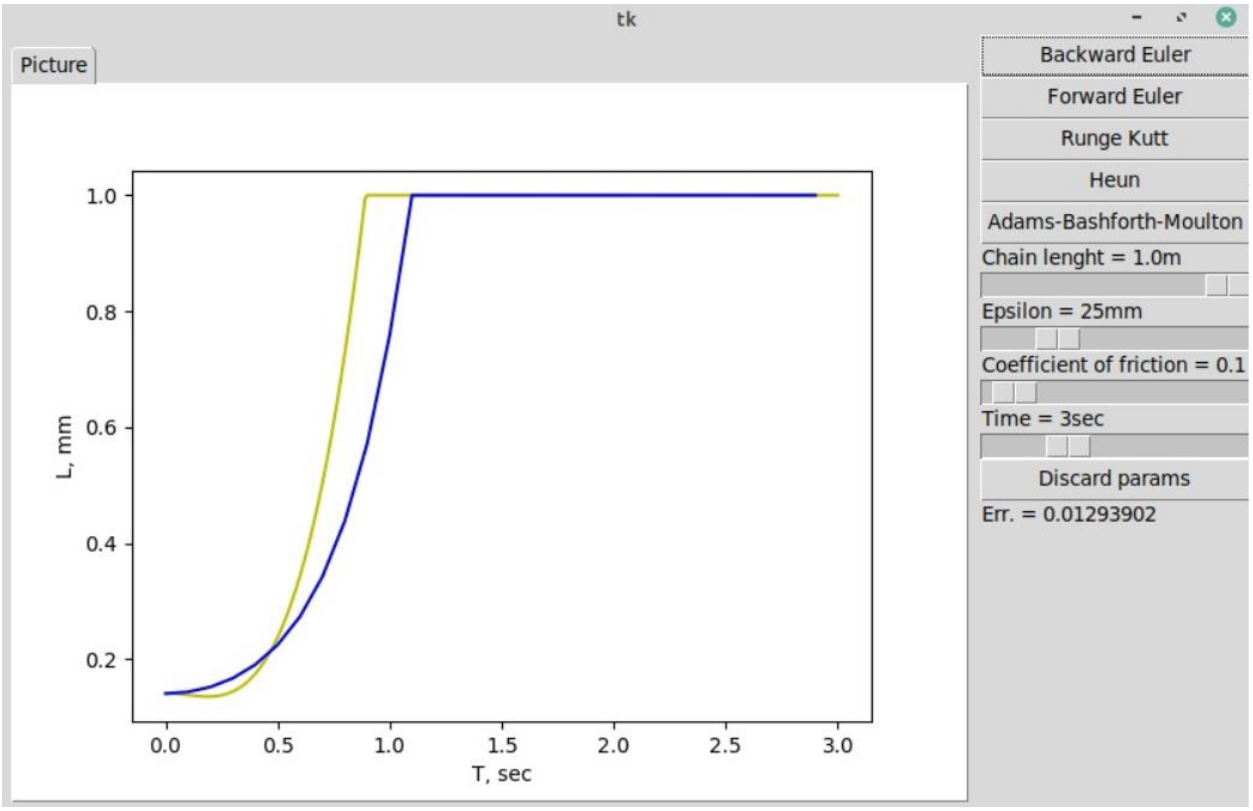
Интерфейс программы включает в себя 6 кнопок, 3 слайдера, окно для графиков и лэйбл с выводом информации об успешном/неуспешном запуске программы:

Первые пять кнопок в интерфейсе вызывают численные методы:

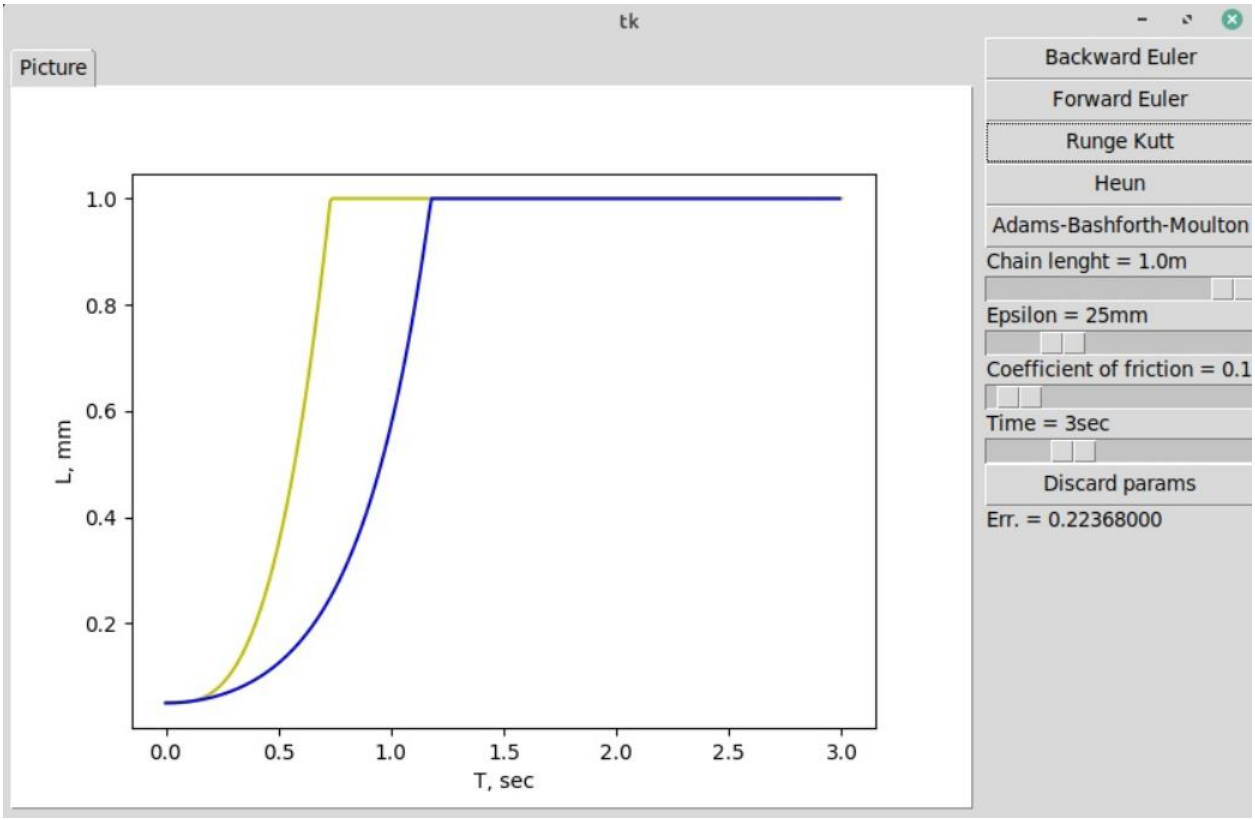
Forward Euler:



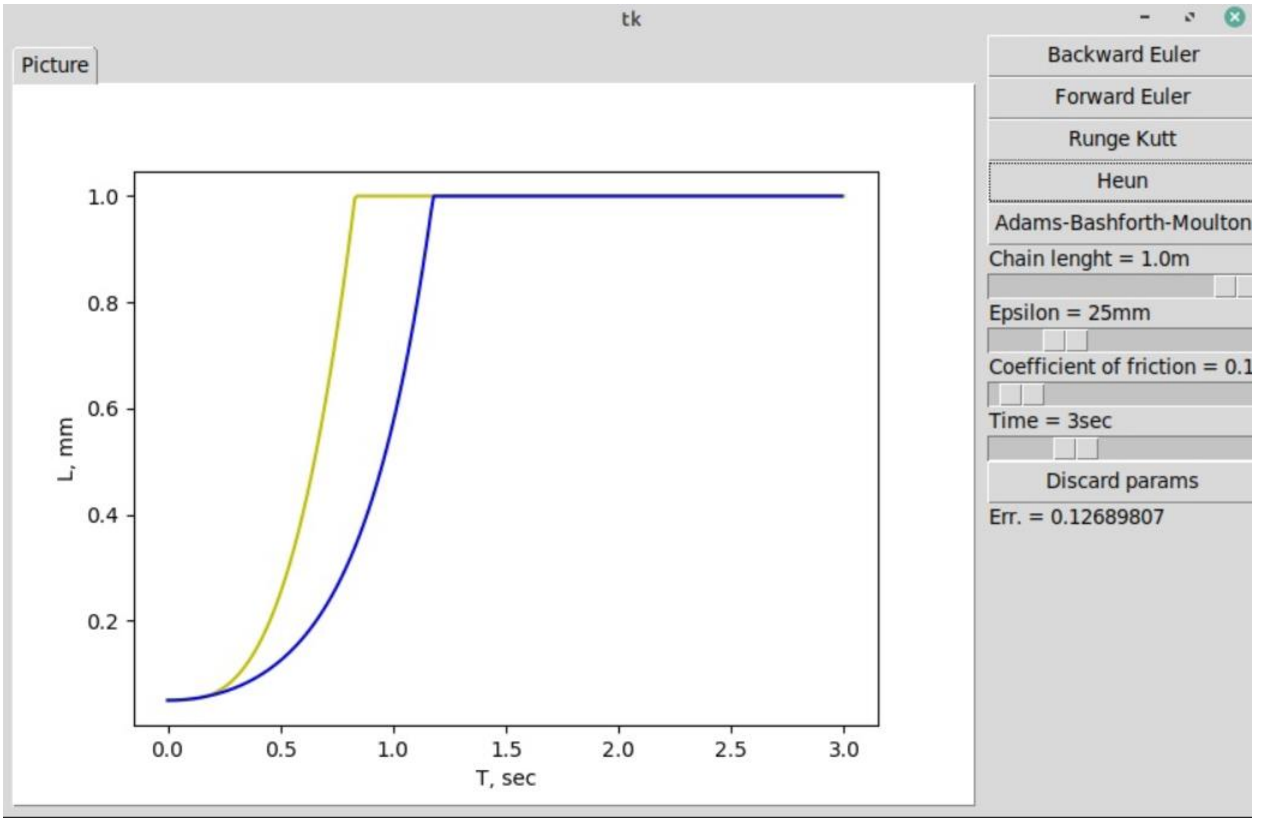
Backward Euler:



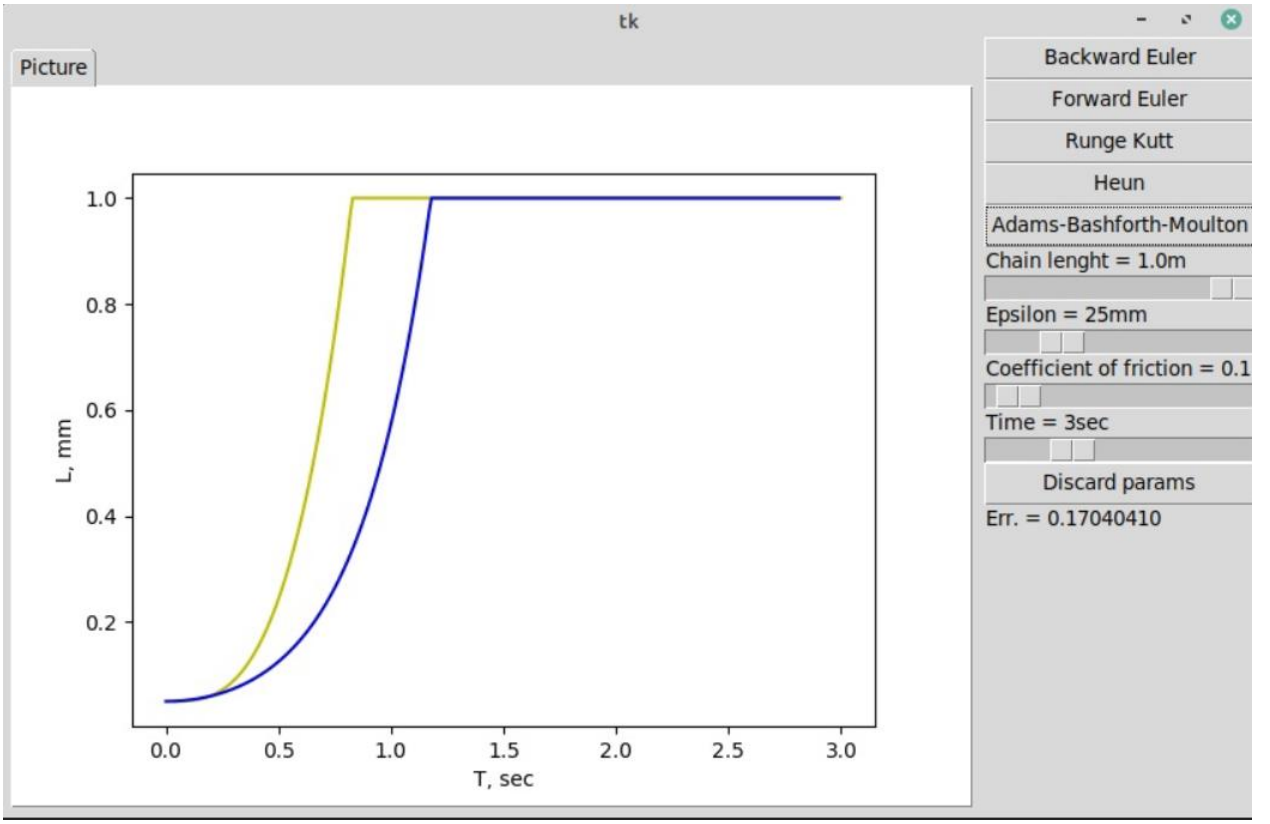
Runge Kutt:



Heun:



Adams-Bashfourth:



С помощью слайдеров можно менять условия поставленной задачи.

Chain length отвечает за длину цепочки, Epsilon за смещение цепочки относительно точки равновесия, Time – за время, кнопка Discard params – сбрасывает значения на исходные.

Также под графиком выводится глобальная ошибка для конкретного метода с шагом h = 0.001

Вывод

В курсовой работе был рассмотрен процесс соскальзывания цепочки. 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 и сравнивающее решение с аналитическим. 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('T, sec')
        self.axes.set_ylabel('L, mm')
        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_chain_lenght = ttk.Label(input_frame)
        self.slider_chain_lenght = ttk.Scale(input_frame, from_ = 0, to_ = 5,
                            command=lambda x:
                            label_chain_lenght.config(text = "Chain lenght = " +
                            str("{0:.1f}".format(self.slider_chain_lenght.get()))
+ "m"))
        self.slider_chain_lenght.set(1)
        label_chain_lenght.config(text = "Chain lenght = " +
str("{0:.1f}".format(self.slider_chain_lenght.get())) + "m")

        label_epsilon = ttk.Label(input_frame)
        self.slider_epsilon = ttk.Scale(input_frame, from_ = 1, to_ = 500,
                            command=lambda x:
                            label_epsilon.config(text = "Epsilon = " +
str(int(self.slider_epsilon.get())) + "mm"))
        self.slider_epsilon.set(25)
        label_epsilon.config(text = "Epsilon = " +
str(int(self.slider_epsilon.get())) + "mm")

        label_coef = ttk.Label(input_frame)
        self.slider_coef = ttk.Scale(input_frame, from_ = 0, to_ = 2,
                            command=lambda x:
                            label_coef.config(text = "Coefficient of friction = "
+
```

```
                        str("{0:.1f}".format(self.slider_coef.get())))
        self.slider_coef.set(0.1)
        label_coef.config(text = "Coefficient of friction = " +
str("{0:.1f}".format(self.slider_coef.get())))

        label_time = ttk.Label(input_frame)
        self.slider_time = ttk.Scale(input_frame, from_ = 0.1, to_ = 10,
                        command=lambda x:
                        label_time.config(text = "Time = " +
                        str("{0:.1f}".format(self.slider_time.get())) +
"sec"))
        self.slider_time.set(3)
        label_time.config(text = "Time = " + str(int(self.slider_time.get())) +
"sec")

        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_chain_lenght.grid(column=0, row=5, columnspan=2, sticky='ew')
        self.slider_chain_lenght.grid(column=0, row=6, columnspan=2, sticky='ew')
        label_epsilon.grid(column=0, row=7, columnspan=2, sticky='ew')
        self.slider_epsilon.grid(column=0, row=8, columnspan=2, sticky='ew')
        label_coef.grid(column=0, row=9, columnspan=2, sticky='ew')
        self.slider_coef.grid(column=0, row=10, columnspan=2, sticky='ew')
        label_time.grid(column=0, row=11, columnspan=2, sticky='ew')
        self.slider_time.grid(column=0, row=12, columnspan=2, sticky='ew')
        button_discard_param.grid(column=0, row=13, columnspan=2, sticky='ew')
        self.label_err_msg.grid(column=0, row=14, columnspan=2, sticky='ew')

    def button_BE_clicked(self):
        self.plot.draw_lists(backward_euler.BackwardEuler(
            0.01, self.slider_coef.get(), self.slider_chain_lenght.get(),
            self.slider_epsilon.get(), self.slider_time.get()
            ).execute())

    def button_FE_clicked(self):
        self.plot.draw_lists(forward_euler.ForwardEuler(
            0.01, self.slider_coef.get(), self.slider_chain_lenght.get(),
            self.slider_epsilon.get(), self.slider_time.get()
            ).execute())

    def button_RK_clicked(self):
        self.plot.draw_lists(runge_kutta.Runge_Kutt(
            0.01, int(self.slider_coef.get()), self.slider_chain_lenght.get(),
            self.slider_epsilon.get(), self.slider_time.get()
            ).execute())

    def button_H_clicked(self):
        self.plot.draw_lists(heun.Heun(
```

```
            0.01, int(self.slider_coef.get()), self.slider_chain_lenght.get(),
            self.slider_epsilon.get(), self.slider_time.get()
            ).execute())

    def button_ADM_clicked(self):
        self.plot.draw_lists(adams_bashforth_moulton.ABM(
            0.01, int(self.slider_coef.get()), self.slider_chain_lenght.get(),
            self.slider_epsilon.get(), self.slider_time.get()
            ).execute())

    def button_discard_param_clicked(self):
        self.slider_epsilon.set(2 * self.eps)
        self.slider_chain_lenght.set(100)
        self.slider_coef.set(200)
        self.check_sliders()

    def check_sliders(self):
        # print("hi")
        if (self.slider_chain_lenght.get() <= self.slider_epsilon.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 = 0.01, _coef = 0.1, _chain_len = 1, _eps = 25, time_ =
3):
        self.h = _h
        self.coef = _coef
        self.chain_len = _chain_len
        self.eps = _eps
        self.x = np.array([0.0])
        self.time = time_
        self.f = open('tmp.txt', 'w')

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

    def RungeKutta4thOrder(self, x):
        appr = int((self.time - 0)/self.h)

        x = 0
        y = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

        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(appr):
            yp2 = x + mf.myFunc(x, self.coef, self.chain_len)*(self.h/2)
            yp3 = x + mf.myFunc(yp2, self.coef, self.chain_len)*(self.h/2)
            yp4 = x + mf.myFunc(yp3, self.coef, self.chain_len)*self.h
            y = y + (self.h/6)*(mf.myFunc(x, self.coef, self.chain_len) +
2*mf.myFunc(yp2, self.coef, self.chain_len) + 2*mf.myFunc(yp3, self.coef,
self.chain_len) + mf.myFunc(yp4, self.coef, self.chain_len))
```

```
        if (y > 1):
            y = 1

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

        y_res = np.append(y_res, y)

    return [xsol, y_res]



def ABM4thOrder(self):
    dx = int((self.time - 0) / self.h)

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

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

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

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

    for i in range(3, dx):
        y0prime = mf.myFunc(y[i], self.coef, self.chain_len)
        y1prime = mf.myFunc(y[i - 1], self.coef, self.chain_len)
        y2prime = mf.myFunc(y[i - 2], self.coef, self.chain_len)
```

```
            y3prime = mf.myFunc(y[i - 3], self.coef, self.chain_len)

            ypredictor = y[i] + (self.h/24)*(55*y0prime - 59*y1prime + 37*y2prime
- 9*y3prime)
            ypp = mf.myFunc(ypredictor, self.coef, self.chain_len)

            yn = y[i] + (self.h/24)*(9*ypp + 19*y0prime - 5*y1prime + y2prime)

            if (yn > 1):
                yn = 1
            # print (yn)

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

            self.x = xsol

            y_res = np.append(y_res, yn)
        return [xsol, y_res]

    def execute(self):
        [ts, ys] = self.ABM4thOrder()
        print(len(ts))

        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.time, self.h)
        for index in t:
            self.f.write(str(index) + ' ')
```

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


        for i in t:
            T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 /
self.chain_len) * i) + 2 * self.eps/2000 * np.exp(-np.sqrt((1 + self.coef) * 9.8
/ self.chain_len) * i) + self.coef * self.chain_len / (1 + self.coef)
            if (T > 1):
                T = 1
            self.f.write(str(T) + ' ')
        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 = 0.01, _coef = 0.1, _chain_len = 1, _eps = 25, time_ =
3):
        self.h = _h
        self.coef = _coef
        self.chain_len = _chain_len
        self.eps = _eps
        self.time = time_
        self.f = open('tmp.txt', 'w')


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


    def main_BE(self):
        appr = int((self.time - 0)/self.h)
```

```
    j = 0

    x = 0
    y = {}
    y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        F_x_t = mf.myFunc(x, self.coef, self.chain_len)/(1+self.h)

        j += 1
        y[j] = y[j-1] + self.h*F_x_t
        print(y[j])

        x += self.h
        if (y[j] > 1):
            y[j] = 1
        self.f.write(str(x) + ' ')
    self.f.write('\n')

    return y

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

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

    t = np.arange(0, self.time, self.coef)

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

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


        for i in t:
            T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 /
self.chain_len) * i) + 2 * self.eps/2000 * np.exp(-np.sqrt((1 + self.coef) * 9.8
/ self.chain_len) * i) + self.coef * self.chain_len / (1 + self.coef)
            if (T > 1):
                T = 1
            self.f.write(str(T) + ' ')
        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 = 0.01, _coef = 0.1, _chain_len = 1, _eps = 25, time_ =
3):
        self.h = _h
        self.coef = _coef
        self.chain_len = _chain_len
        self.eps = _eps
        self.time = time_
        self.f = open('tmp.txt', 'w')


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


    def main_FE(self):
        appr = int((self.time - 0)/self.h)
```

```
    j = 0

    x = 0
    y = {}
    y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000

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

    for i in range(appr):
        F_x_t = mf.myFunc(x, self.coef, self.chain_len)

        j += 1
        y[j] = y[j-1] + self.h*F_x_t
        print (y[j])

        x += self.h
        if (y[j] > 1):
            y[j] = 1
        self.f.write(str(x) + ' ')
    self.f.write('\n')

    return y

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

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

    t = np.arange(0, self.time, self.coef)

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

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


        for i in t:
            T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 /
self.chain_len) * i) + 2 * self.eps/2000 * np.exp(-np.sqrt((1 + self.coef) * 9.8
/ self.chain_len) * i) + self.coef * self.chain_len / (1 + self.coef)
            if (T > 1):
                T = 1
            self.f.write(str(T) + ' ')
        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 = 0.01, _coef = 0.1, _chain_len = 1, _eps = 25, time_ =
3):
        self.h = _h
        self.coef = _coef
        self.chain_len = _chain_len
        self.eps = _eps
        self.time = time_
        self.f = open('tmp.txt', 'w')


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


    def main_H(self):
        appr = int((self.time - 0)/self.h)
```

```
        j = 0


        x = 0
        y = {}
        y[j] = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000


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


        for i in range(appr):
            j += 1
            y[j] = y[j-1] + (self.h/2)*mf.myFunc(x, self.coef, self.chain_len) +
(self.h/2)*mf.myFunc(x + mf.myFunc(x, self.coef, self.chain_len) * self.h,
self.coef, self.chain_len)
            print(y[j])
            if (y[j] > 1):
                y[j] = 1


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


        return y


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


        t = np.arange(0, self.time, self.h)
        for index in t:
            self.f.write(str(index) + ' ')
        self.f.write('\n')
```

```
        for i in t:
            T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 /
self.chain_len) * i) + 2 * self.eps/2000 * np.exp(-np.sqrt((1 + self.coef) * 9.8
/ self.chain_len) * i) + self.coef * self.chain_len / (1 + self.coef)
            if (T > 1):
                T = 1
            self.f.write(str(T) + ' ')
        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 = 0.01, _coef = 0.1, _chain_len = 1, _eps = 25,
time_ = 3):
        self.h = _h
        self.coef = _coef
        self.chain_len = _chain_len
        self.eps = _eps
        self.time = time_
        self.f = open('tmp.txt', 'w')


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


    def RKF45(self):
        appr = int((self.time - 0)/self.h)


        j = 0
```

```
                x = 0

                y = {}

                y[j] = self.coef * 1 / (1 + self.coef) + 2 * self.eps/1000


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


                for i in range(appr):

                    yp2 = x + mf.myFunc(x, self.coef, self.chain_len)*(self.h/5)

                    yp3 = x + mf.myFunc(x, self.coef,
self.chain_len)*(3*self.h/40) + mf.myFunc(yp2, self.coef,
self.chain_len)*(9*self.h/40)

                    yp4 = x + mf.myFunc(x, self.coef,
self.chain_len)*(3*self.h/10) - mf.myFunc(yp2, self.coef,
self.chain_len)*(9*self.h/10) + mf.myFunc(yp3, self.coef,
self.chain_len)*(6*self.h/5)

                    yp5 = x - mf.myFunc(x, self.coef,
self.chain_len)*(11*self.h/54) + mf.myFunc(yp2, self.coef,
self.chain_len)*(5*self.h/2) - mf.myFunc(yp3, self.coef,
self.chain_len)*(70*self.h/27) + mf.myFunc(yp4, self.coef,
self.chain_len)*(35*self.h/27)

                    yp6 = x + mf.myFunc(x, self.coef,
self.chain_len)*(1631*self.h/55296) + mf.myFunc(yp2, self.coef,
self.chain_len)*(175*self.h/512) + mf.myFunc(yp3, self.coef,
self.chain_len)*(575*self.h/13824) + mf.myFunc(yp4, self.coef,
self.chain_len)*(44275*self.h/110592) + mf.myFunc(yp5, self.coef,
self.chain_len)*(253*self.h/4096)


                    j += 1

                    y[j] = y[j-1] + self.h*(37*mf.myFunc(x, self.coef,
self.chain_len)/378 + 22 * self.eps*mf.myFunc(yp3, self.coef, self.chain_len)/621
+ 125*mf.myFunc(yp4, self.coef, self.chain_len)/594 + 512*mf.myFunc(yp6,
self.coef, self.chain_len)/1771)


                    x += self.h

                    if (y[j] > 1):

                        y[j] = 1

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

                self.f.write('\n')


                return y
```

```
        def execute(self):
            ys = self.RKF45()
            for index in ys:
                self.f.write(str(ys[index]) + ' ')
            self.f.write('\n')

            t = np.arange(0, self.time, self.h)
            for index in t:
                self.f.write(str(index) + ' ')
            self.f.write('\n')

            for i in t:
                T = 2 * self.eps/2000 * np.exp(np.sqrt((1 + self.coef) * 9.8 /
self.chain_len) * i) + 2 * self.eps/2000 * np.exp(-np.sqrt((1 + self.coef) * 9.8
/ self.chain_len) * i) + self.coef * self.chain_len / (1 + self.coef)
                if (T > 1):
                    T = 1
                self.f.write(str(T) + ' ')
            self.f.write('\n')

            return 0
```

Файл my_func.py:

```
import numpy as np


def myFunc(x, mu, l):
    dy = 4.9 * x**2 * (mu + 1) / l - 9.8 * mu * x
    # dy = 5.self.time9 * (x)**2 - 0.98 * (x)
    return dy
```