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 2^{nd} 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:

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

Отсюда следует, что ускорение будет выглядеть так:

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

После получения скорости v в программе используется формула для равноускоренного движения:

$$v = v0 * t + \frac{at^2}{2}$$

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:

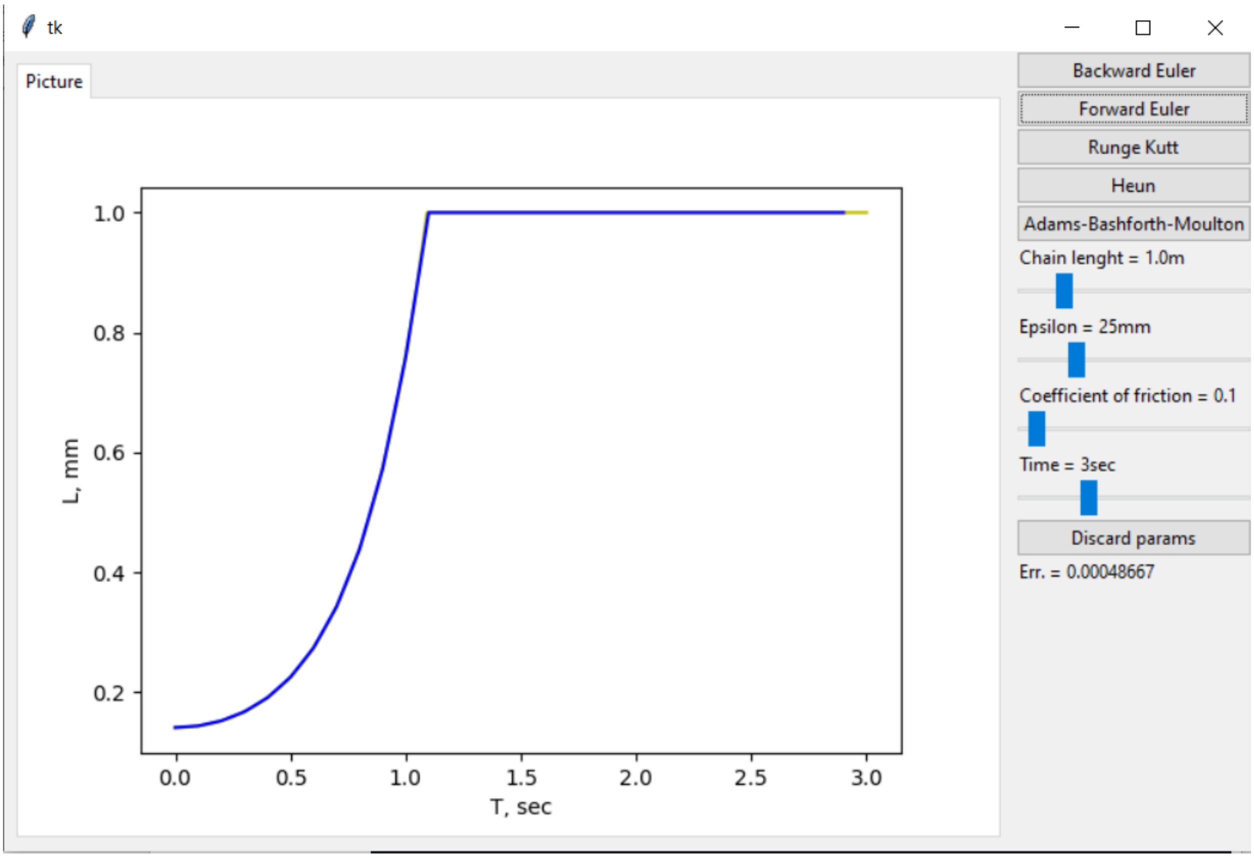
```
def main_FE(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    v = 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):
        a, v = mf.myFunc(x, y[j], v, self.coef, self.chain_len, self.h)

        j += 1
        y[j] = y[j-1] + v * self.h + (a * self.h ** 2) / 2
        print (y[j])

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

    return y
```

a,v – ускорение и скорость, полученные в результате выполнения функции f(x, t), self.h – это длина шага по x, my_func.py выглядит так:

```
import numpy as np


def myFunc(x, v, mu, l):
    # x - длина свисающей части
    a = (1 + mu) * 9.8 * x / l - mu * 9.8
    # v = v + (-9.8 * (x + h) * mu + (9.8 * (x + h) * y * (1 + mu))/l) - (-9.8 * x * mu + (9.8 * x * y * (1 + mu))/l)
    # v += a*h
    return a, v
```

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

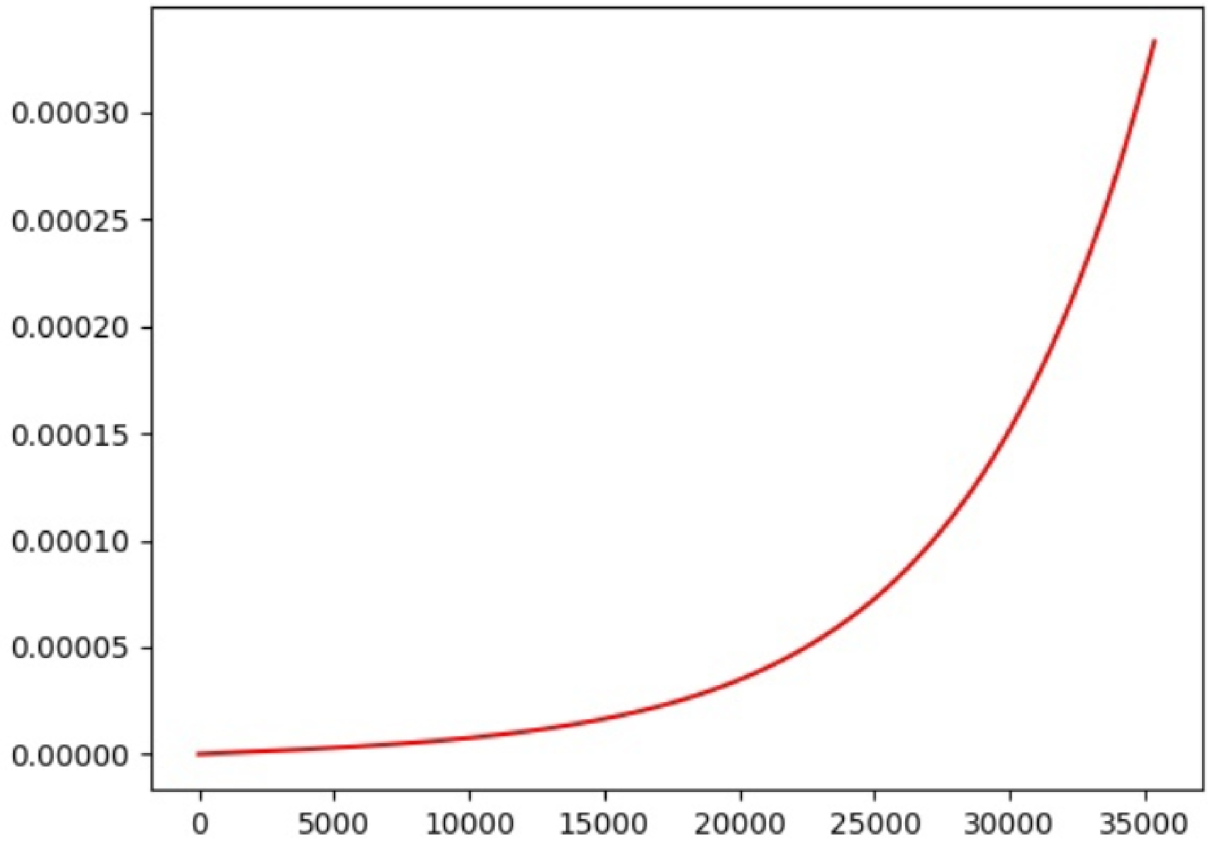
Для сравнения с аналитическим решением использовался график, желтая линия – график метода, синяя – аналитическое решение.



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 средняя глобальная ошибка для шага h = 0.0001 и график глобальной ошибки.


0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
5000
10000
15000
20000
25000
30000
35000

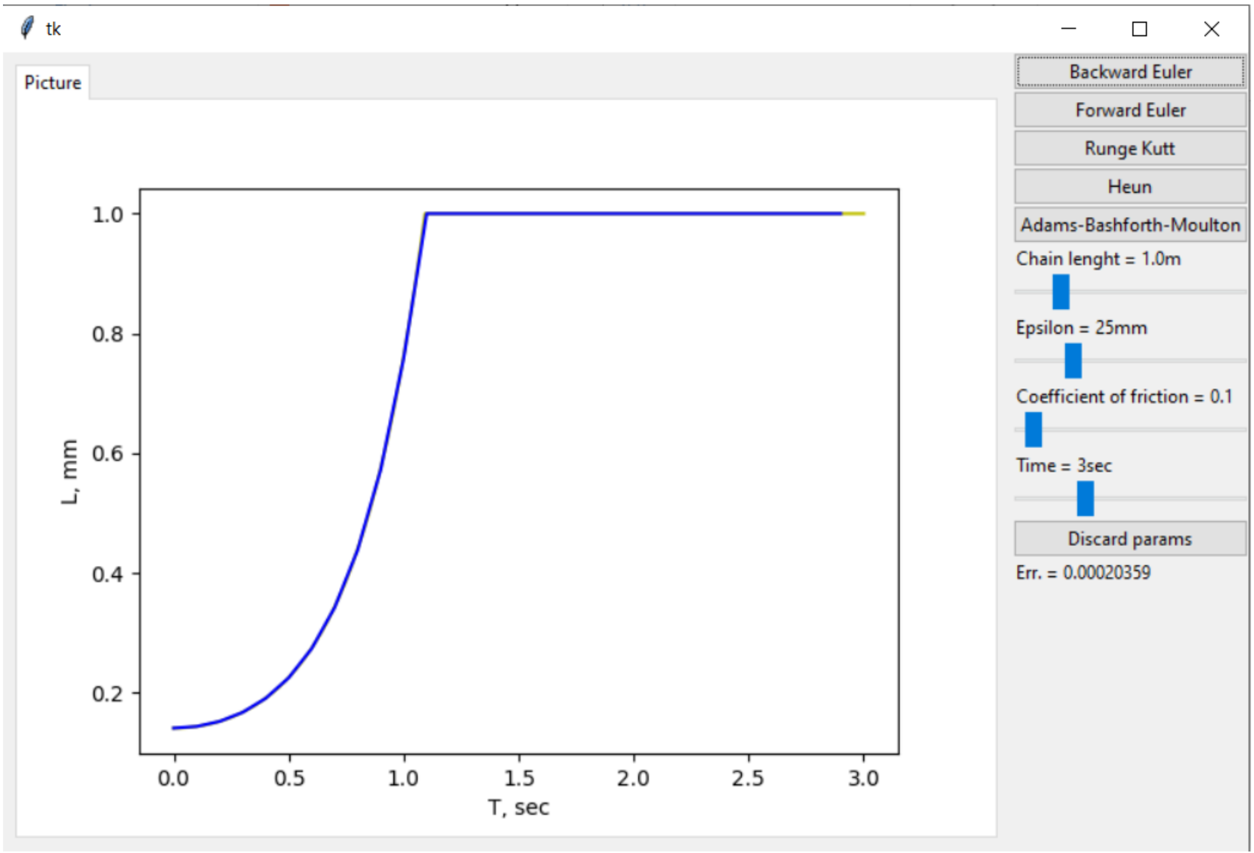
Обратный метод Эйлера

В нашем коде этот метод был реализован так:

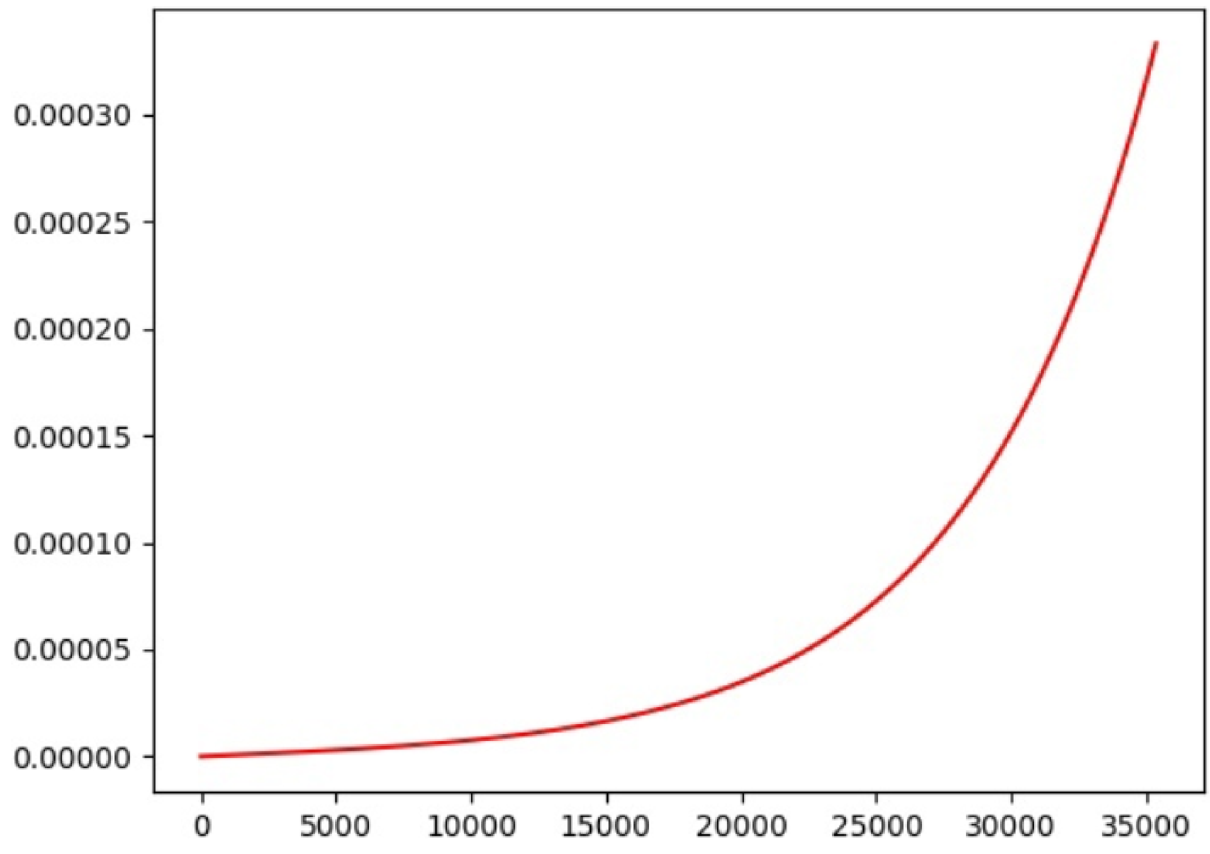
```
def main_BE(self):
    appr = int((self.time - 0)/self.h)

    j = 0

    v = 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):
        a, v = mf.myFunc(x, y[j], v, self.coef, self.chain_len, self.h)

        y[j+1] = y[j] + (v + (a * self.h) / 2)* self.h

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

    return y
```

Где a,v – ускорение и скорость, полученные в результате выполнения функции 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 средняя глобальная ошибка для шага h = 0.0001 и график глобальной ошибки.



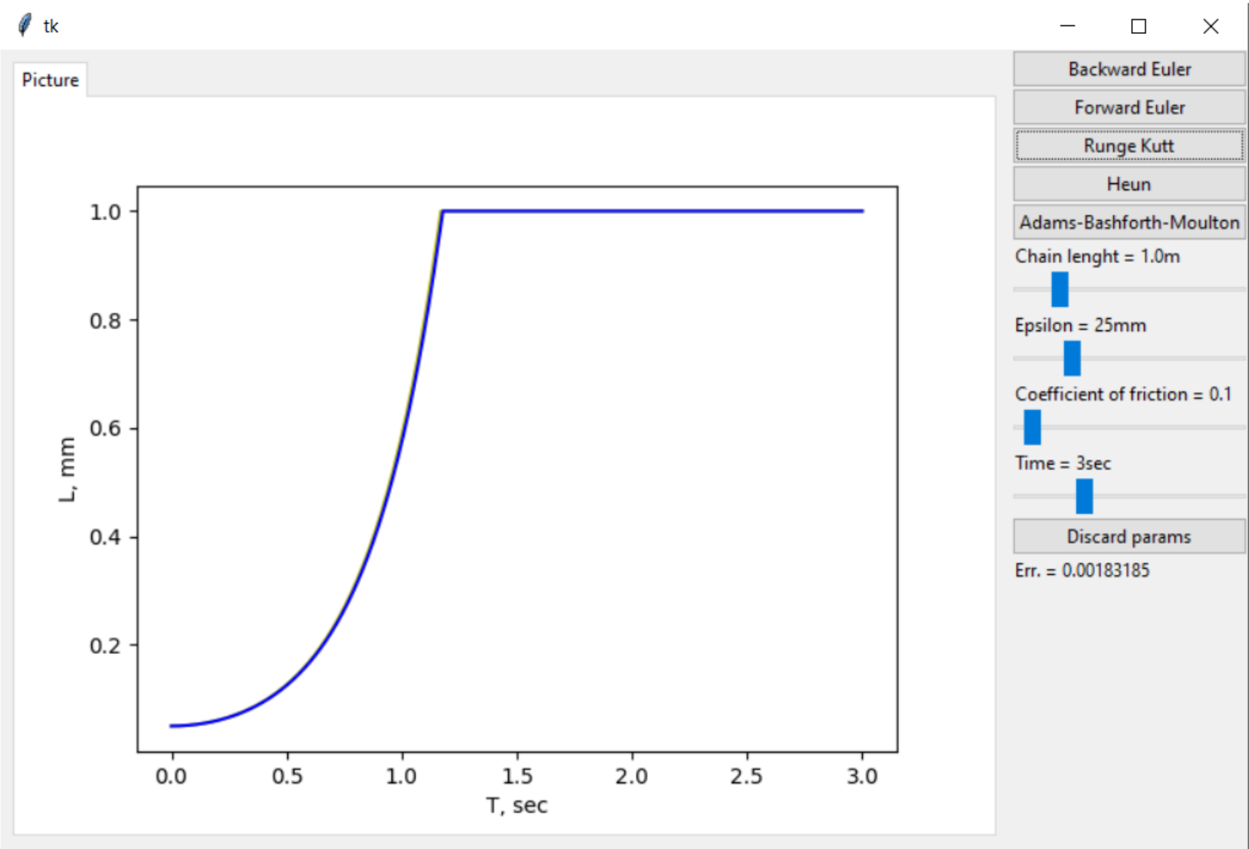
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
5000
10000
15000
20000
25000
30000
35000

Метод Рунге-Кутты-Фельберга 4-5 порядка

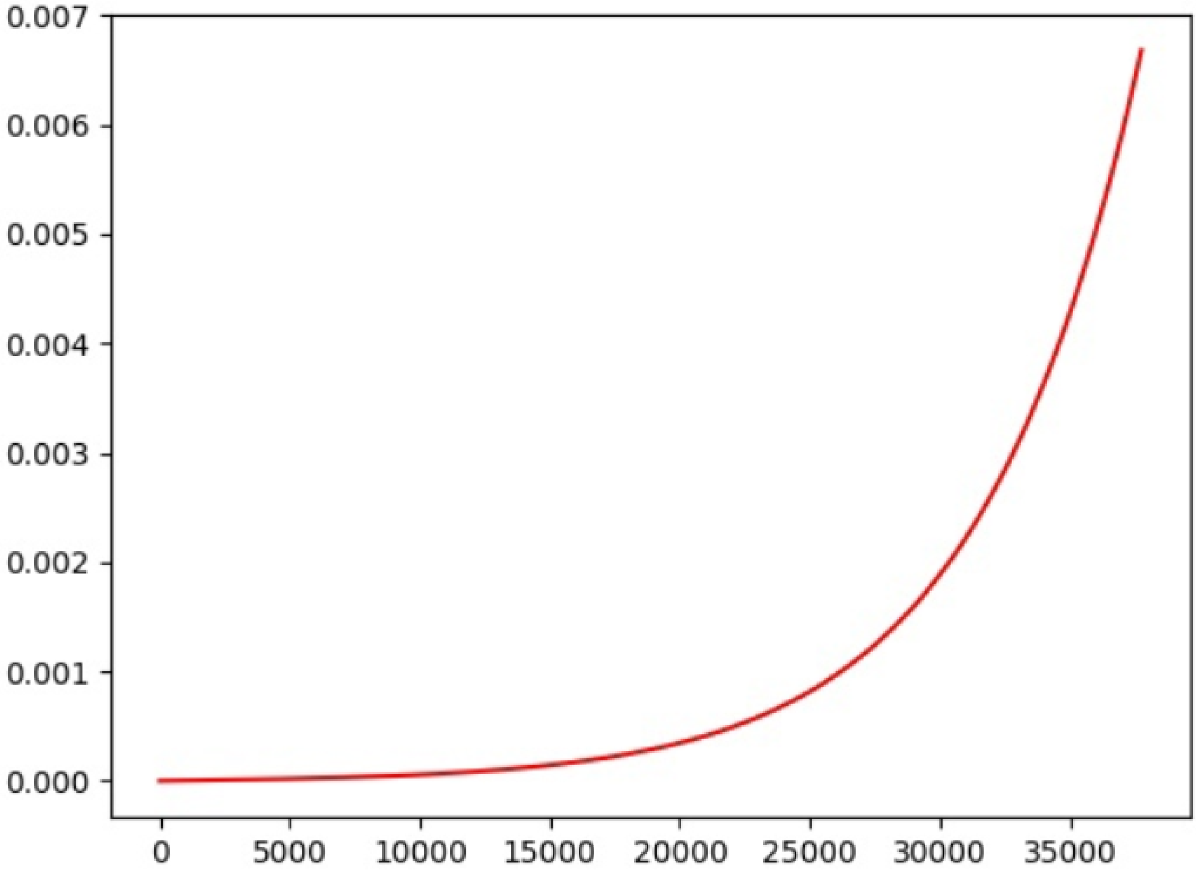
Этот метод был реализован таким образом:

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

    j = 0

    v = 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):
        a1, v1 = mf.myFunc(x, y[j], v, self.coef, self.chain_len, self.h)
        yp2 = y[j] + (v1 + (a1 * self.h/5) / 2)*(self.h/5)
        a2, v2 = mf.myFunc(x + 1/5*self.h, yp2, v1, self.coef, self.chain_len, self.h)
        yp3 = y[j] + (v1 + (3 * a1 * self.h/40) / 2)*(3*self.h/40) + (v2 + (9 * a2 * self.h/40) / 2)*(9*self.h/40)
        a3, v3 = mf.myFunc(x + 3/10*self.h, yp3, v2, self.coef, self.chain_len, self.h)
        yp4 = y[j] + (v1 + (3 * a1 * self.h/10) / 2)*(3*self.h/10) - (v2 + (9 * a2 * self.h/10) / 2)*(9*self.h/10) + (v3 + (6 * a3 * self.h/5) / 2
        a4, v4 = mf.myFunc(x + 3/5*self.h, yp4, v3, self.coef, self.chain_len, self.h)
        yp5 = y[j] - (v1 + (11 * a1 * self.h/54) / 2)*(11*self.h/54) + (v2 + (5 * a2 * self.h/2) / 2)*(5*self.h/2) - (v3 + (70 * a3 * self.h/27) /
        a5, v5 = mf.myFunc(x + self.h, yp5, v4, self.coef, self.chain_len, self.h)
        yp6 = y[j] + (v1 + (1631 * a1 * self.h/55296) / 2)*(1631*self.h/55296) + (v2 + (175 * a2 * self.h/512) / 2)*(175*self.h/512) + (v3 + (575
        a6, v6 = mf.myFunc(x + 7/8*self.h, yp6, v5, self.coef, self.chain_len, self.h)

        j += 1
        y[j] = y[j-1] + self.h*(37*(v1 + (a1 * self.h) / 2)/378 + 250 * (v3 + (a3 * self.h) / 2)/621 + 125*(v4 + (a4 * self.h) / 2)/594 + 512*(v6

        v = v1
        x += self.h
        if (y[j] > 1):
            y[j] = 1
```

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



Также после выполнения метода, на экран выводится его средняя глобальная ошибка для шага h = 0.0001 и график глобальной ошибки.

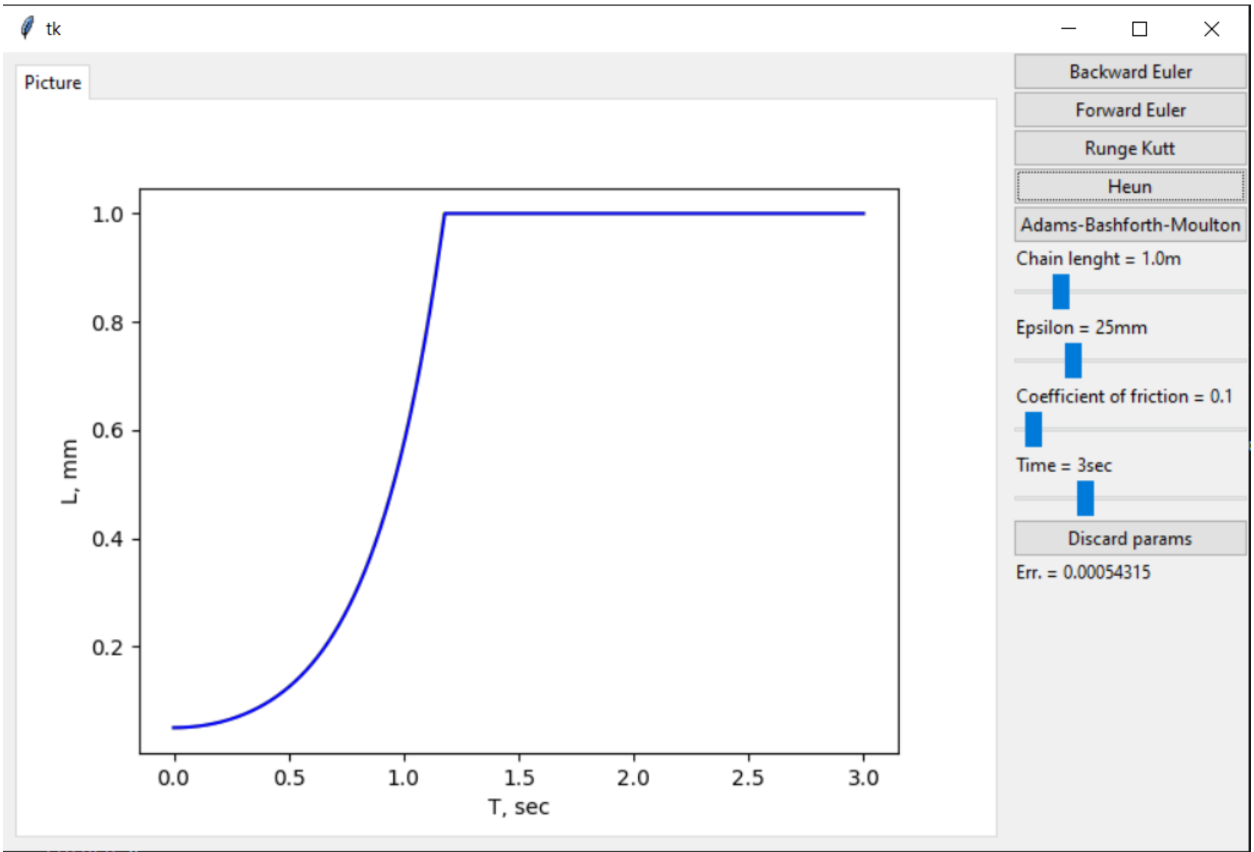

0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
5000
10000
15000
20000
25000
30000
35000

Метод Хойна

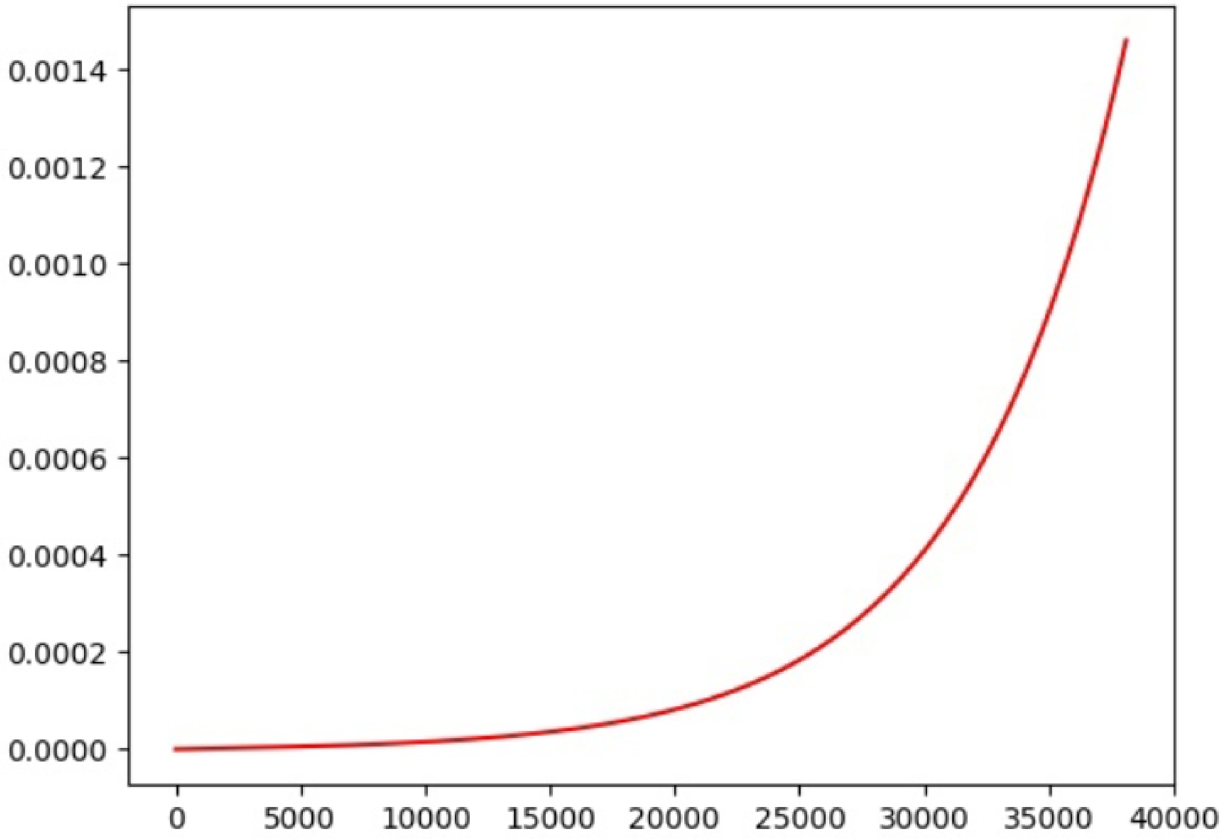
В работе метод был реализован так:

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

    j = 0

    v = 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):
        a1, v1 = mf.myFunc(x, y[j], v, self.coef, self.chain_len, self.h)
        a2, v2 = mf.myFunc(x + (v1 * self.h + (a1 * self.h ** 2) / 2) * self.h, y[j], v1, self.coef, self.chain_len, self.h)

        j += 1
        y[j] = y[j-1] + (v1 * self.h / 2 + (a1 * (self.h/2) ** 2) / 2) + (v2 * self.h/2 + (a2 * (self.h/2) ** 2) / 2)

        v = v1
        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 средняя глобальная ошибка для шага h = 0.0001 и график глобальной ошибки.

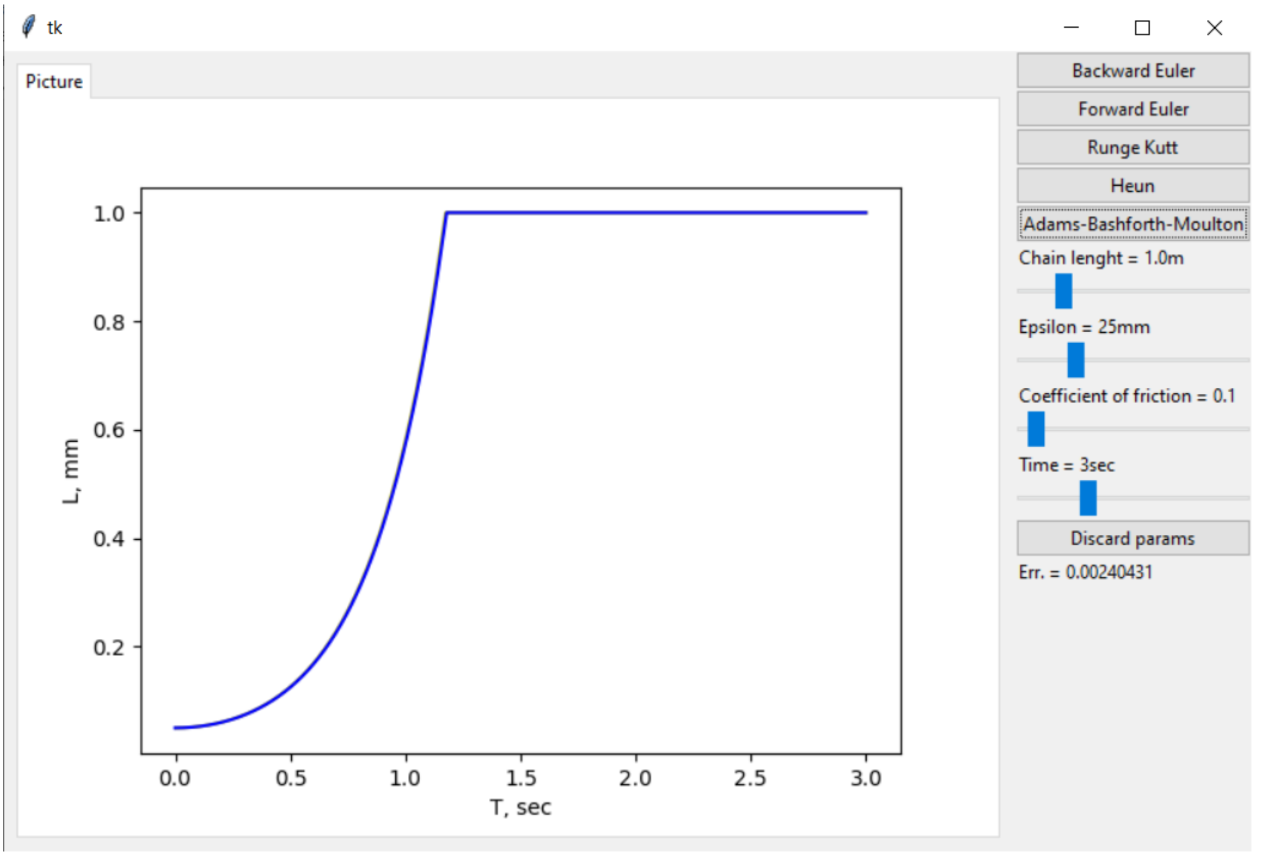

0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0
5000
10000
15000
20000
25000
30000
35000
40000

Метод Адамса-Башфорта

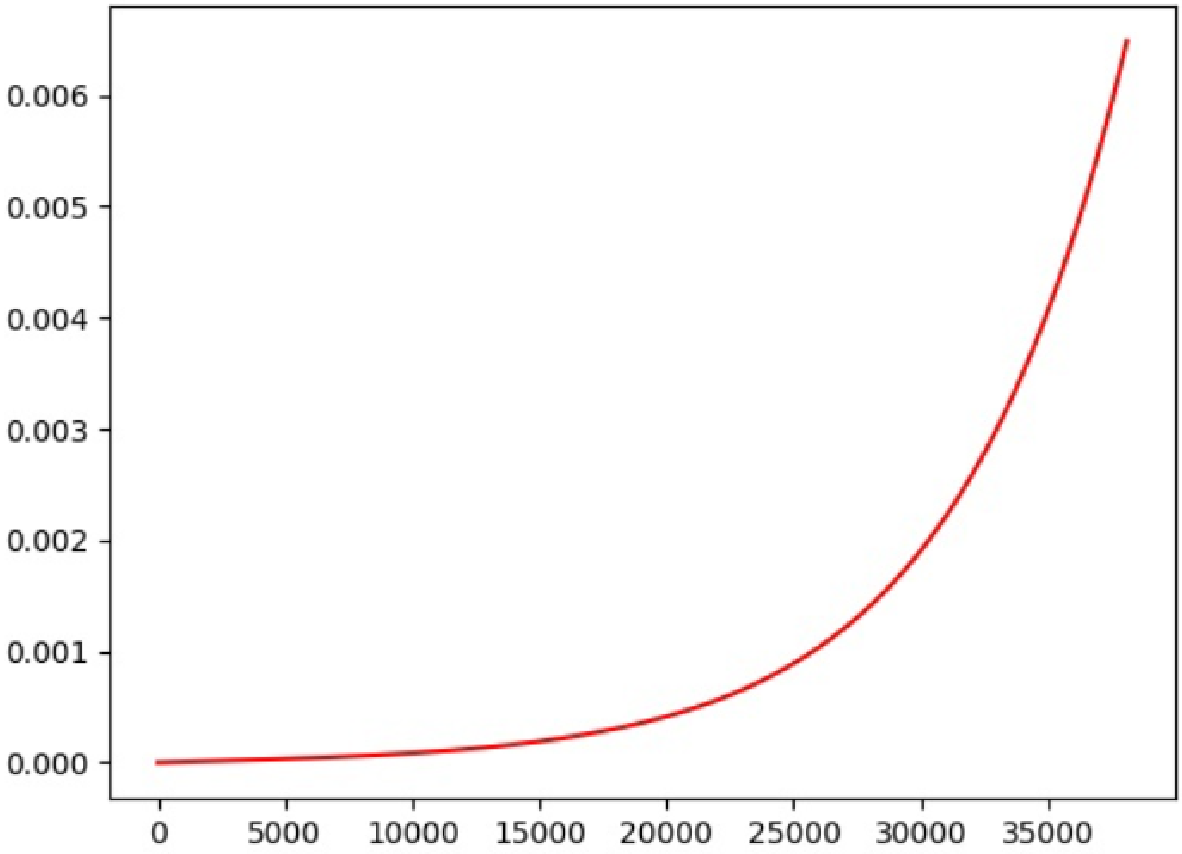
В работе метод был реализован так:

```
    y = yy
    yn = yy[0]
    y_res = np.empty(0)

    V = 0
    xs = 0

    for i in range(3, dx):
        A1, V1 = mf.myFunc(xs, y[i], V, self.coef, self.chain_len, self.h)
        A2, V2 = mf.myFunc(xs, y[i - 1], V1, self.coef, self.chain_len, self.h)
        A3, V3 = mf.myFunc(xs, y[i - 2], V2, self.coef, self.chain_len, self.h)
        A4, V4 = mf.myFunc(xs, y[i - 3], V3, self.coef, self.chain_len, self.h)

        ypredictor = y[i] + (self.h/24)*(55*(V1 + (A1 * self.h/24) / 2) - 59*(V2 + (A2 * self.h/24) / 2) + 37*(V3 + (A3 * self.h/24) / 2) - 9*(V
        Ap, Vp = mf.myFunc(xs, ypredictor, V, self.coef, self.chain_len, self.h)

        yn = y[i] + (self.h/24)*(9*(Vp + (Ap * self.h/24) / 2) + 19*(V1 + (A1 * self.h/24) / 2) - 5*(V2 + (A2 * self.h/24) / 2) + (V3 + (A3 * se

        if (yn > 1):
            yn = 1
        # print (yn)
        V = V1

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

        self.x = xsol

        y_res = np.append(y_res, yn)
```

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



Также после выполнения метода, на экран выводится его средняя глобальная ошибка для шага h = 0.0001 и график глобальной ошибки.


0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
5000
10000
15000
20000
25000
30000
35000

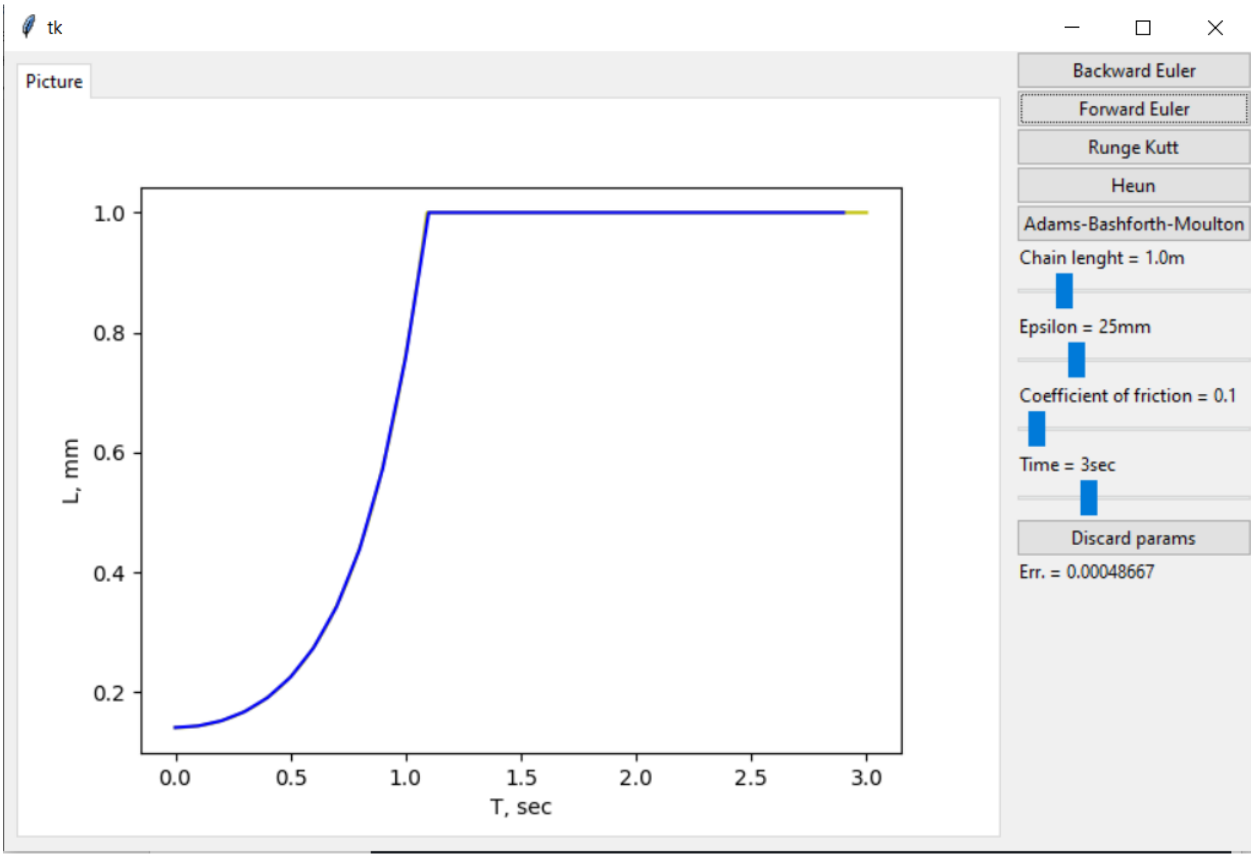
GUI

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

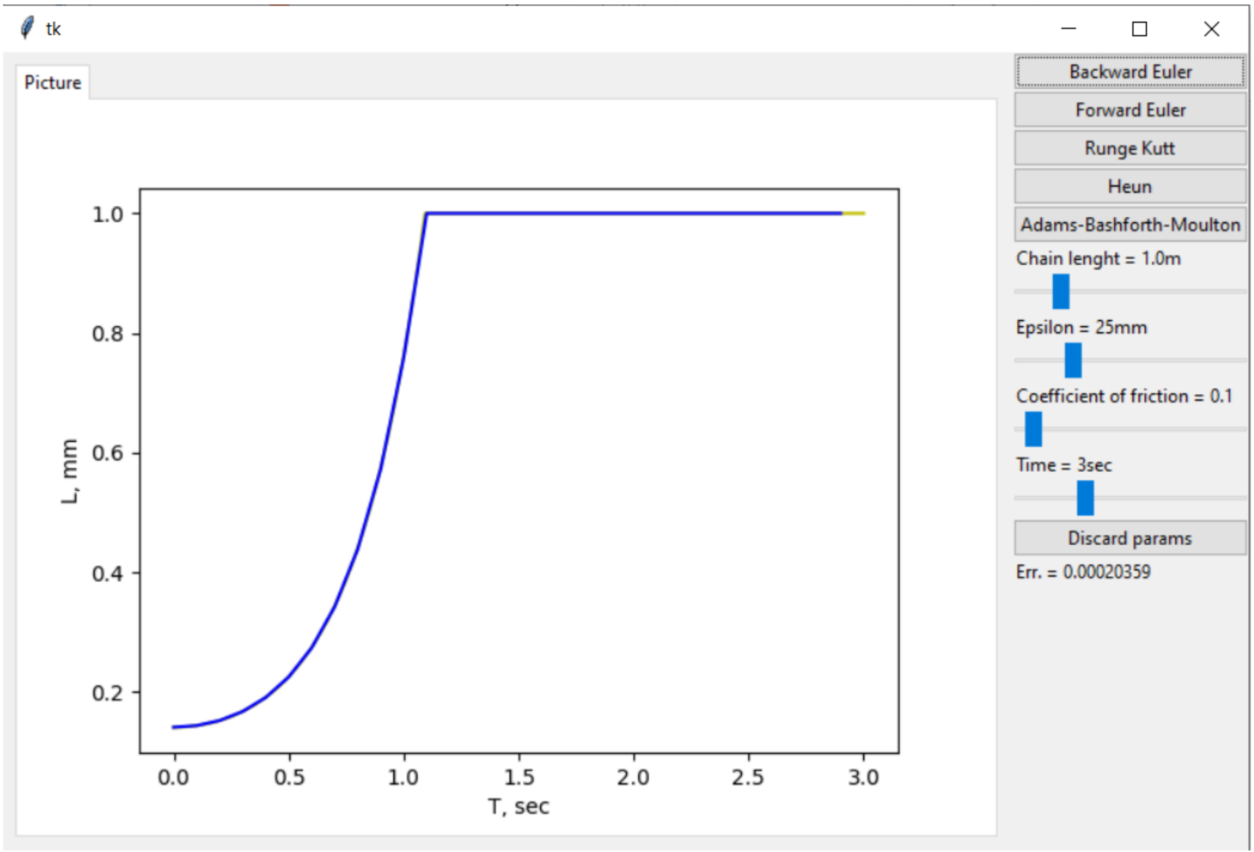
Интерфейс программы включает в себя 6 кнопок, 3 слайдера, окно для графиков и лэйбл с выводом информации об успешном/неуспешном запуске программы:

Первые пять кнопок в интерфейсе вызывают численные методы:

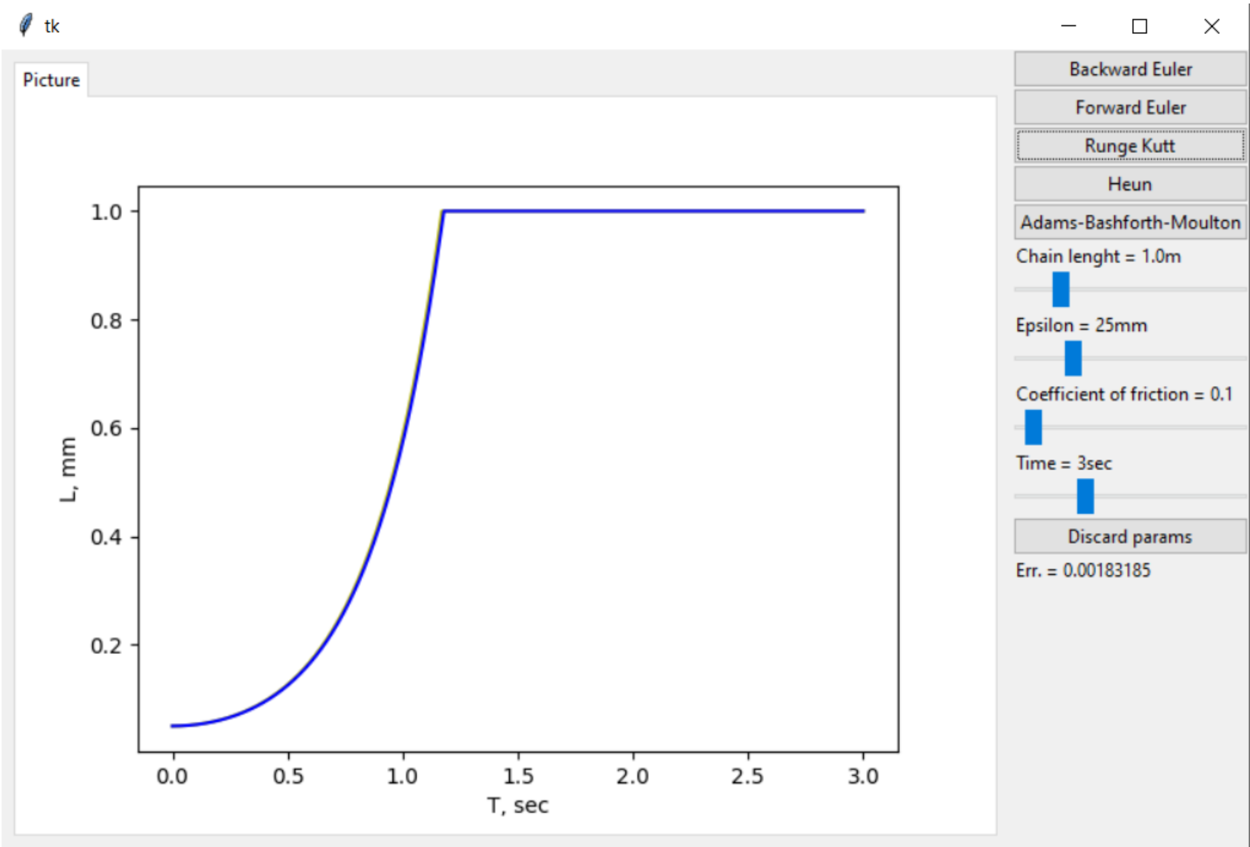
Forward Euler:



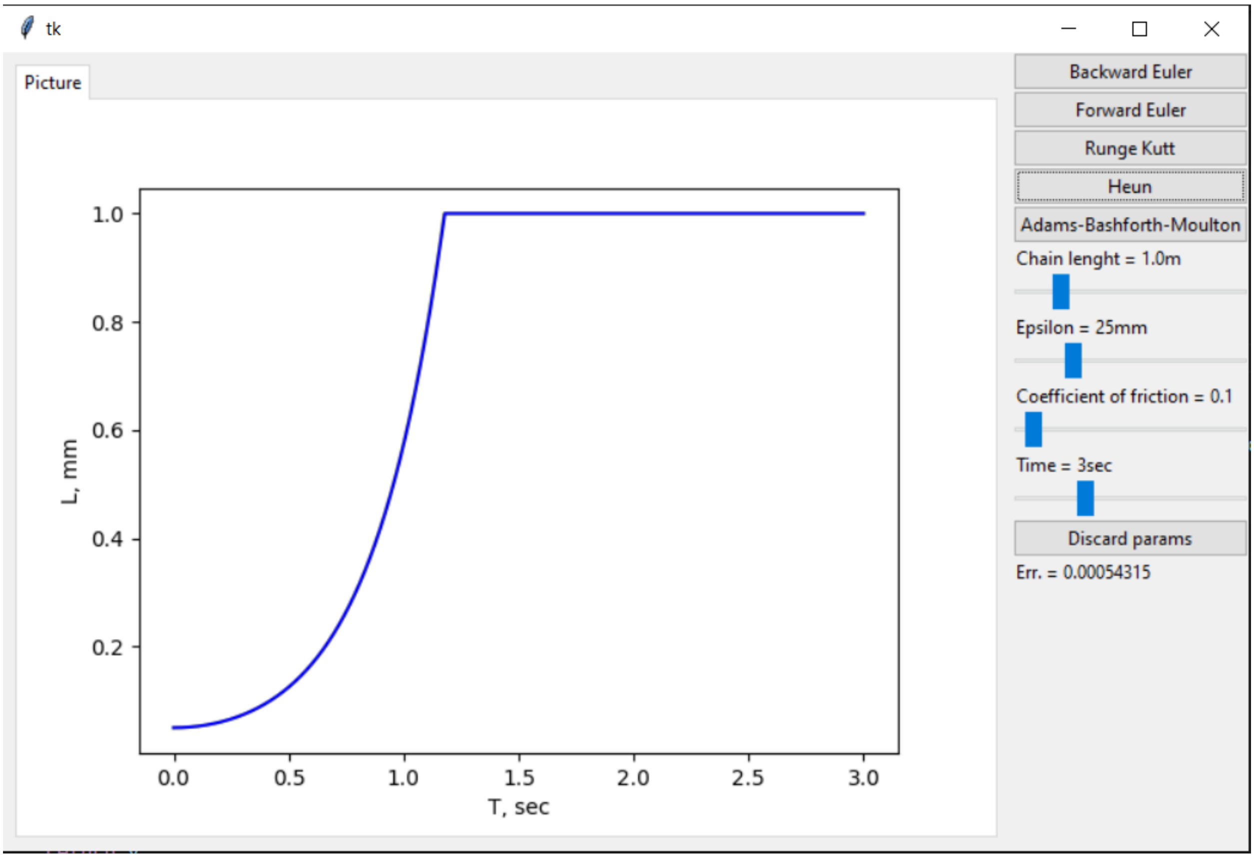
Backward Euler:



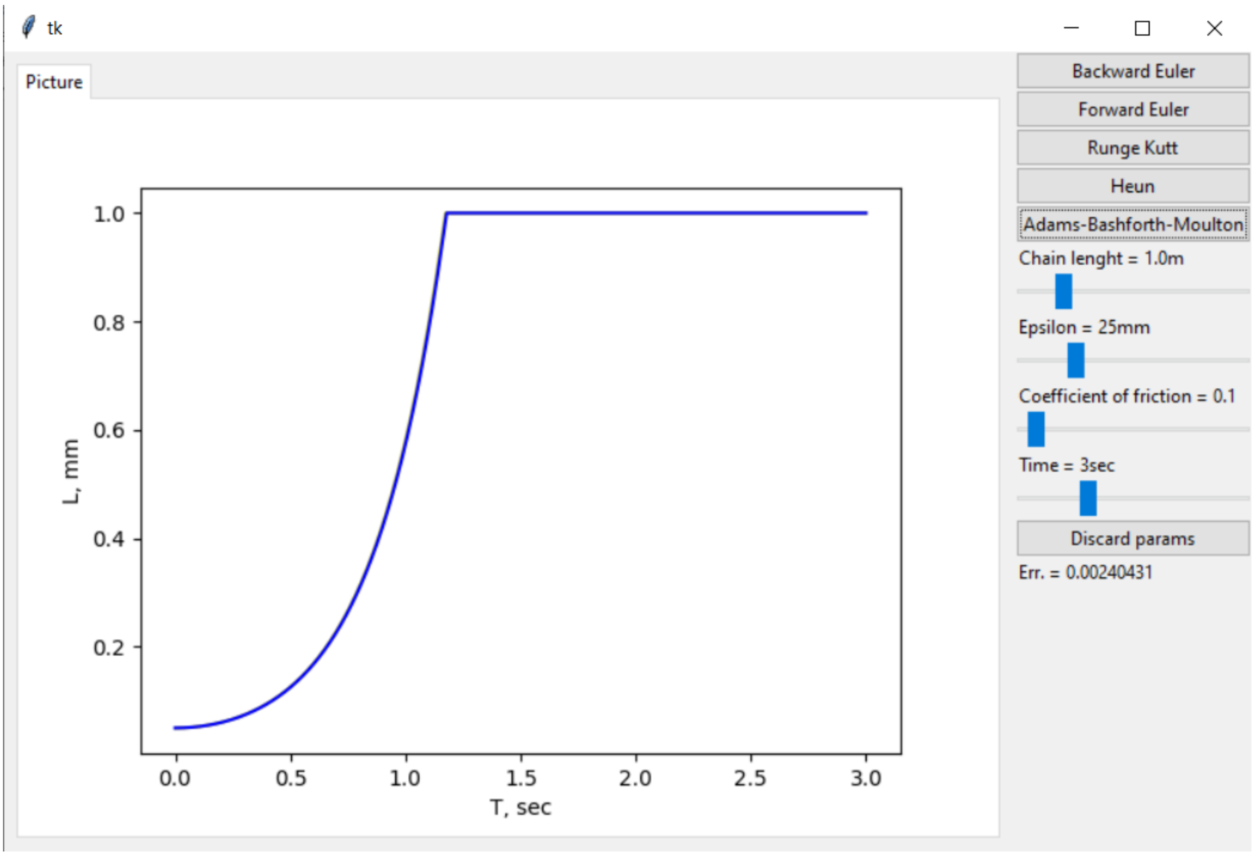
Runge Kutt:



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 справа от графика выводится глобальная ошибка для конкретного метода с шагом h = 0.0001 и график глобальной ошибки.

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/