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 Кэша-Карпа без оценки ошибки и подбора шага», «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 Кэша-Карпа без оценки ошибки и подбора шага», «Метод Хойна», «Метод Адамса 4-го порядка». 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", " Cash-Karp method without error estimation and step selection ", "Heun's method", "Adams method of the 4th order". 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 Кэша-Карпа без оценки ошибки и подбора шага», «Метод Хойна», «Метод Адамса Башфорта 4-го порядка».

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:

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

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:

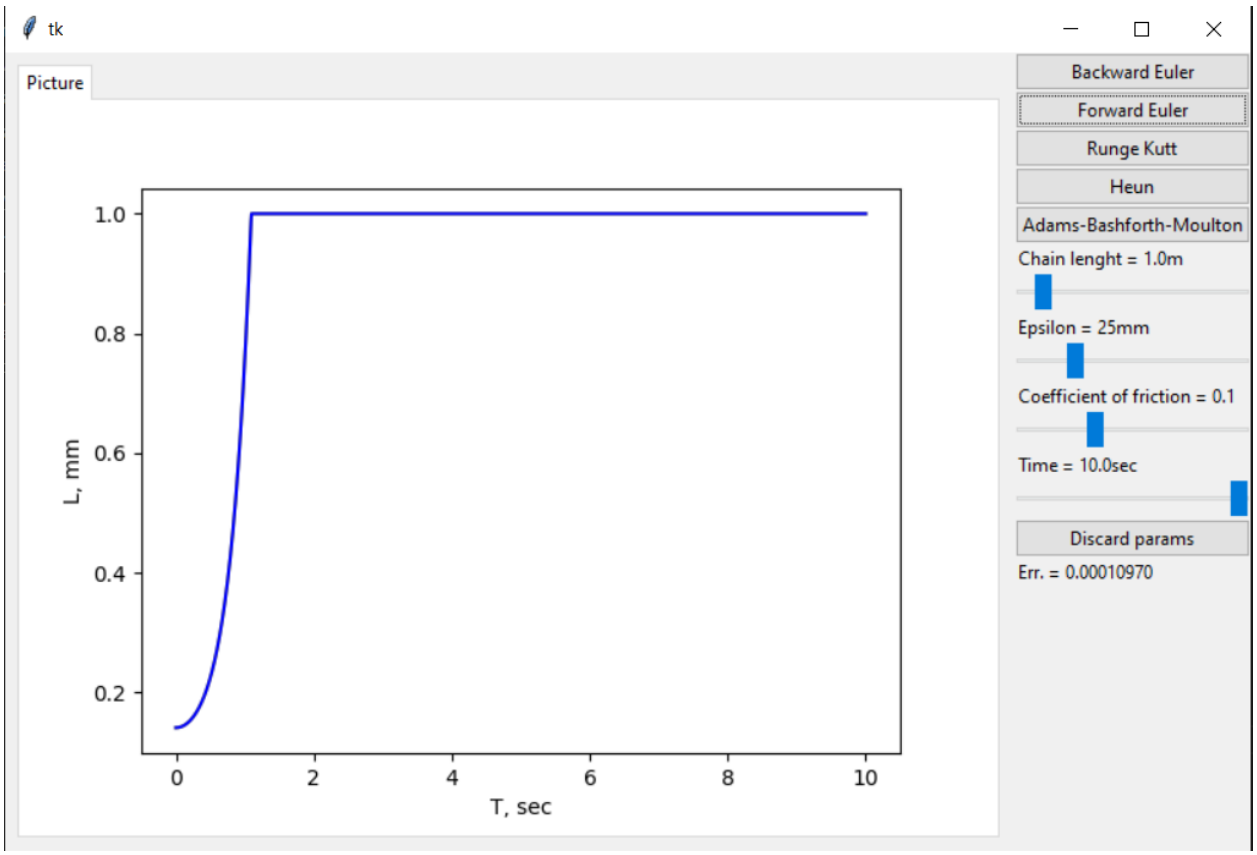
```
def main_FE(self):
    appr = int((self.time - 0)/self.h)

    v = 0
    y0 = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000
    x = np.array([y0, v])
    res = []
    self.f.write(str(0) + ' ')

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

        x = x + xdot * self.h

        if (x[0] > self.chain_len):
            x[0] = self.chain_len
        res = np.append(res, x[0])

        self.f.write(str(i*self.h) + ' ')
    self.f.write('\n')

    return res
```

xdot – ускорение и скорость, полученные в результате выполнения функции f(x, t), self.h – это длина шага по x, self.eps – начальное условие, 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.

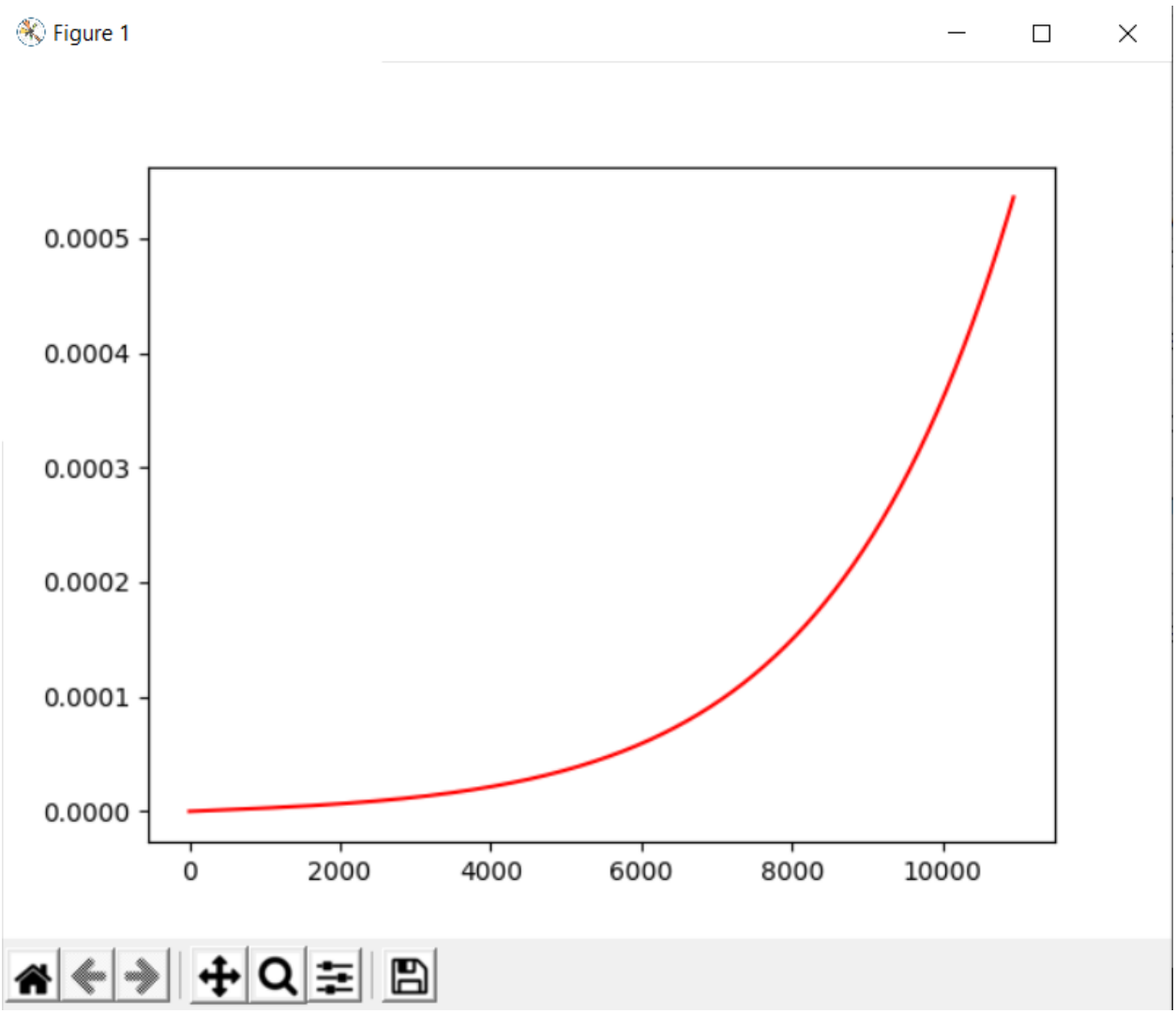


Figure 1
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
10000

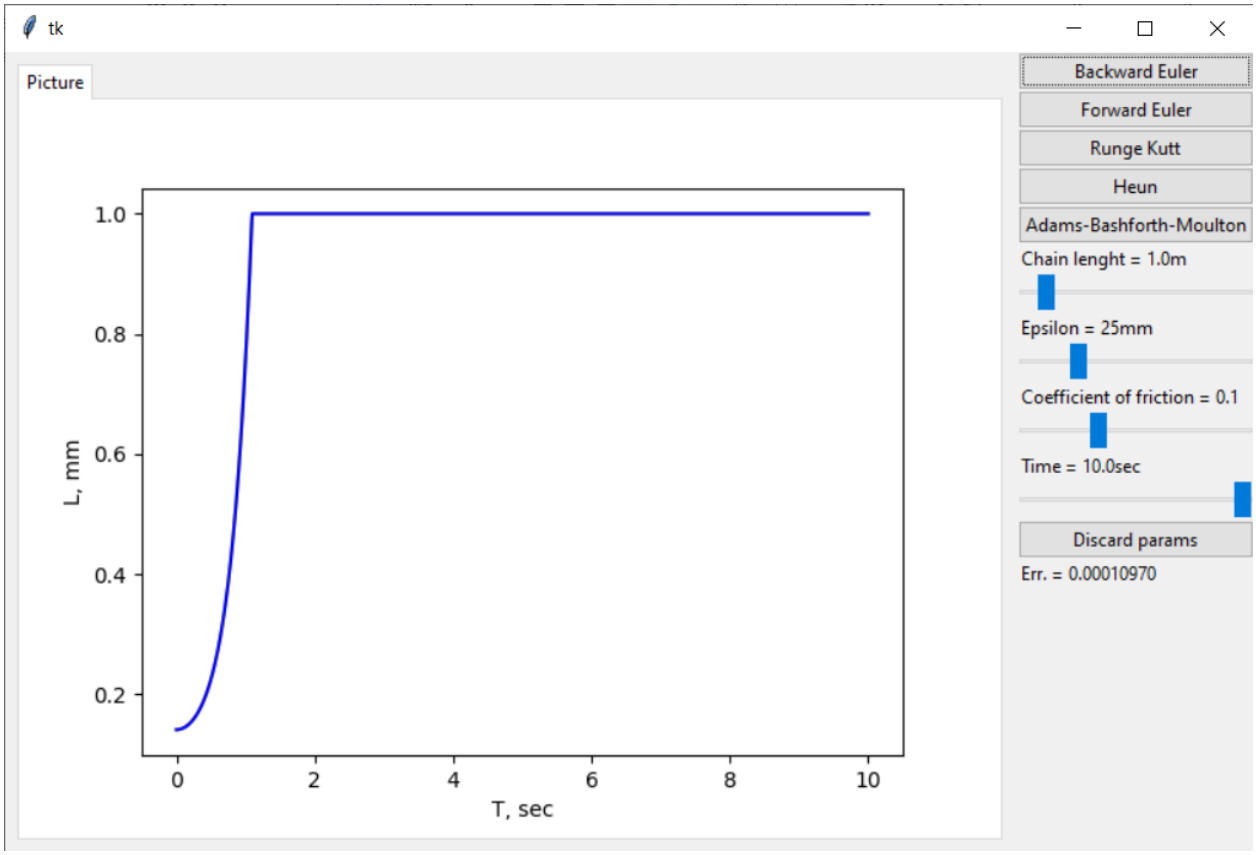
Обратный метод Эйлера

В нашем коде этот метод был реализован так:

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

    v = 0
    y0 = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000
    x = np.array([y0, v])
    res = []

    self.f.write(str(0) + ' ')

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

        x = x + xdot * self.h

        if (x[0] > self.chain_len):
            x[0] = self.chain_len
        res = np.append(res, x[0])
        self.f.write(str(i*self.h) + ' ')
    self.f.write('\n')
    return res
```

Где xdot – 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.

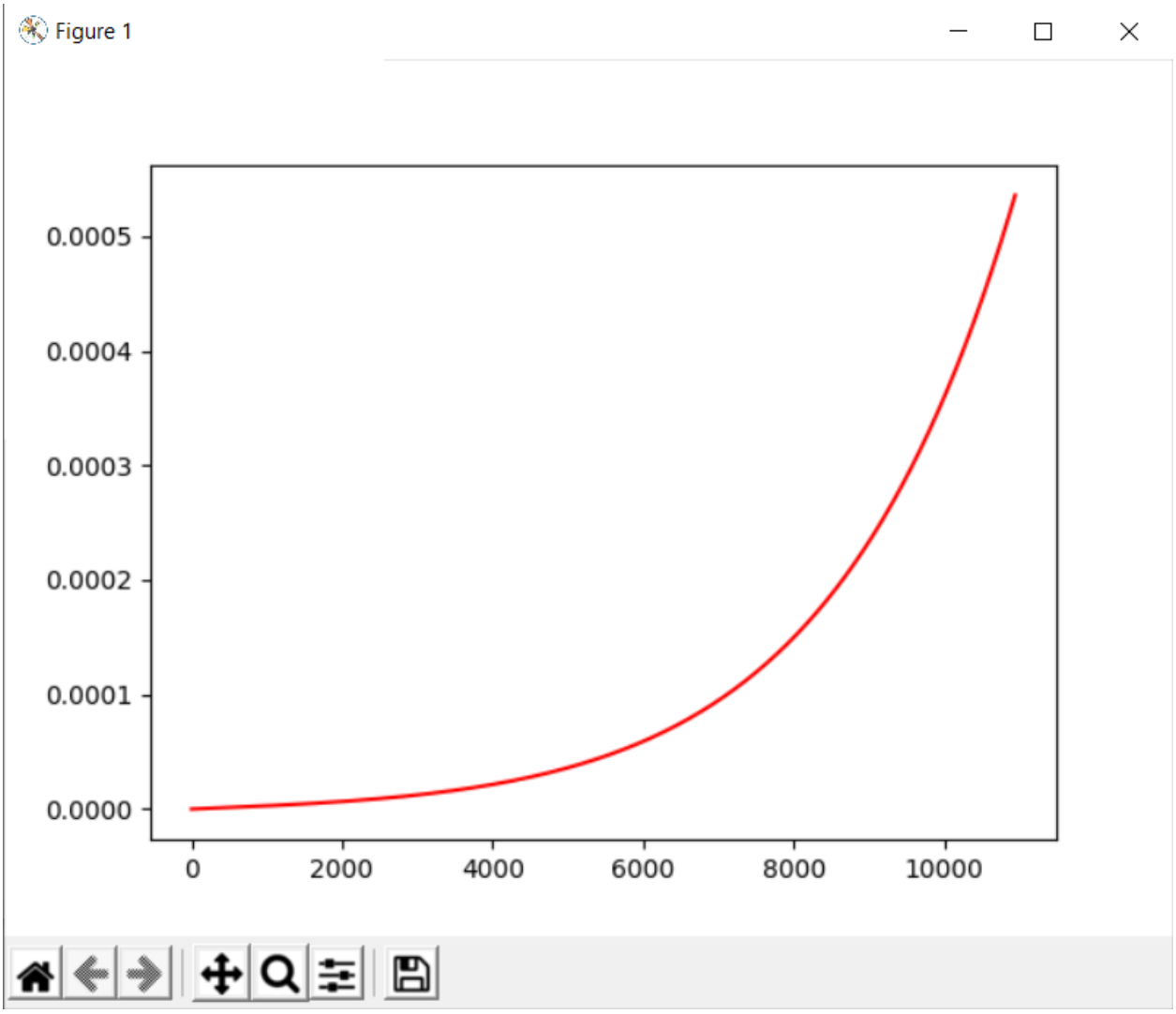


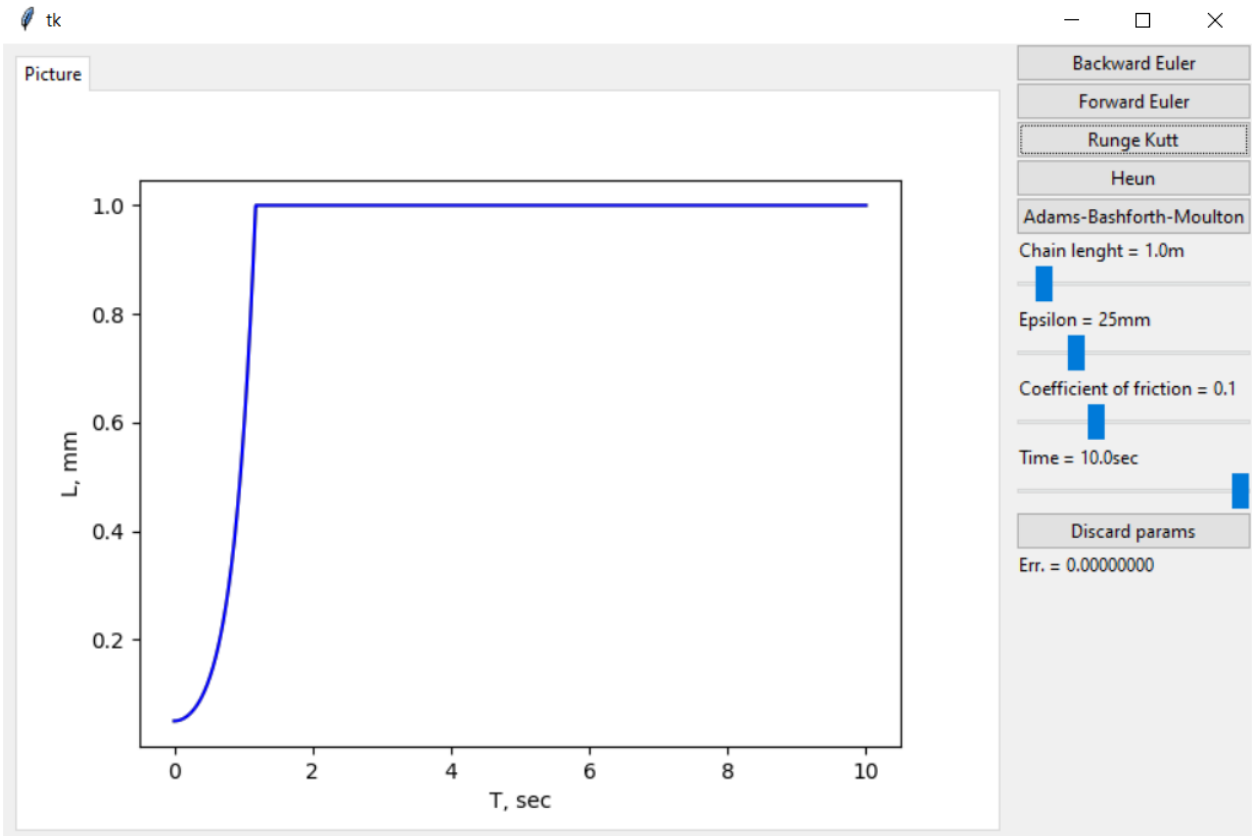
Figure 1
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
10000

Метод Кэша-Карпа без оценки ошибки и подбора шага

В нашем коде этот метод был реализован так:

```
v = 0
y0 = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000
x = np.array([y0, v])
res = []

self.f.write(str(0) + ' ')

for i in range(appr):
    xdot1 = np.array(mf.myFunc(x, self.coef, self.chain_len))
    xp2 = x + xdot1 * self.h/5
    xdot2 = np.array(mf.myFunc(xp2, self.coef, self.chain_len))
    xp3 = x + xdot1 * (3*self.h/40) + xdot2 * (9*self.h/40)
    xdot3 = np.array(mf.myFunc(xp3, self.coef, self.chain_len))
    xp4 = x + xdot1 * (3*self.h/10) - xdot2 * (9*self.h/10) + xdot3 * (6*self.h/5)
    xdot4 = np.array(mf.myFunc(xp4, self.coef, self.chain_len))
    xp5 = x - xdot1 *(11*self.h/54) + xdot2 * (5*self.h/2) - xdot3 * (70*self.h/27) + xdot4 * (35*self.h/27)
    xdot5 = np.array(mf.myFunc(xp5, self.coef, self.chain_len))
    xp6 = x + xdot1 * (1631*self.h/55296) + xdot2 * (175*self.h/512) + xdot3 * (575*self.h/13824) + xdot4 * (44275*self.h/110592) + xdot5 *
    xdot6 = np.array(mf.myFunc(xp6, self.coef, self.chain_len))

    x = x + self.h * (37* xdot1 / 378 + 250 * xdot3 / 621 + 125 * xdot4 / 594 + 512 * xdot6 / 1771)

    if (x[0] > self.chain_len):
        x[0] = self.chain_len
    res = np.append(res, x[0])
    self.f.write(str(i*self.h) + ' ')
self.f.write('\n')

return res
```

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.

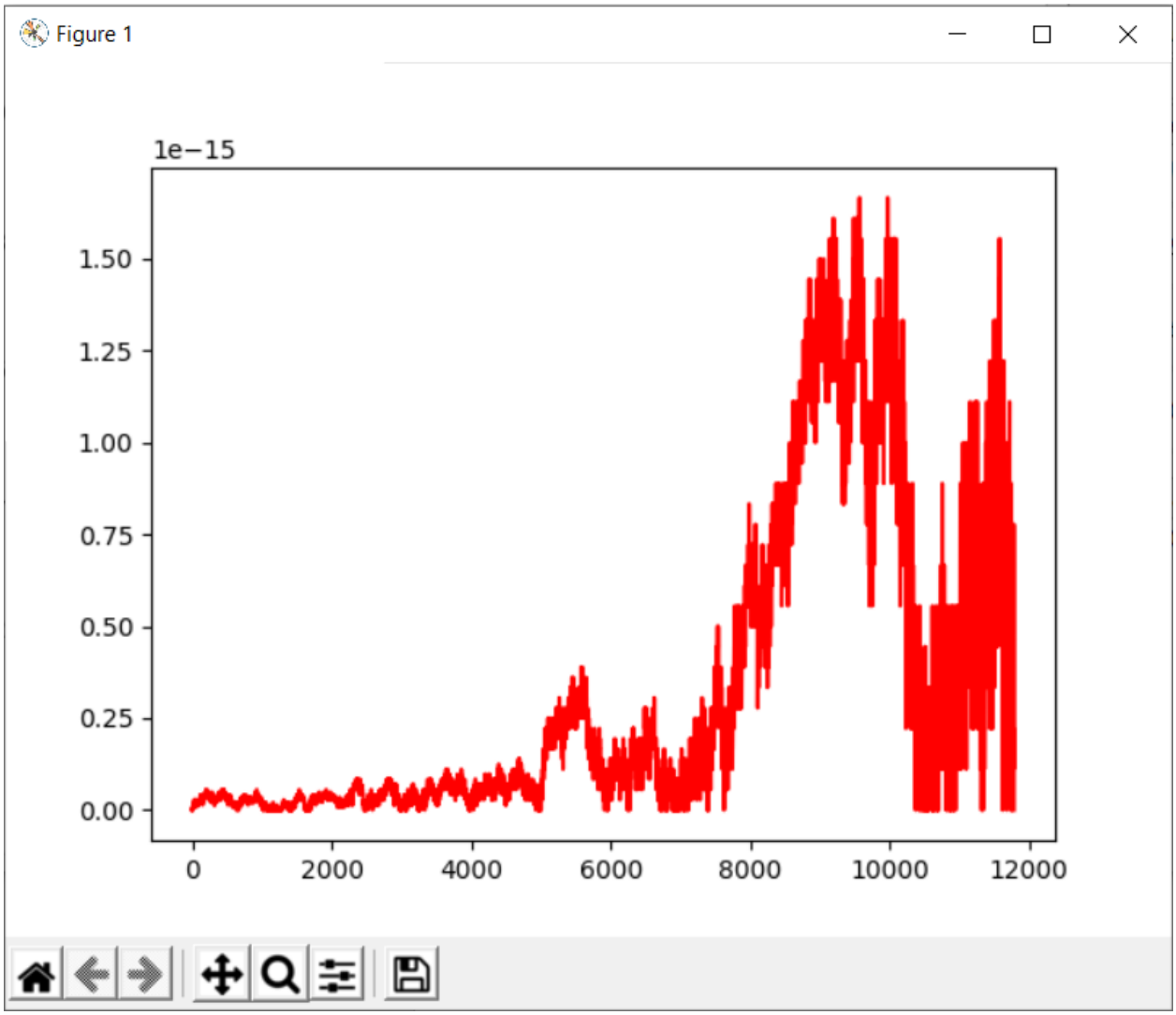


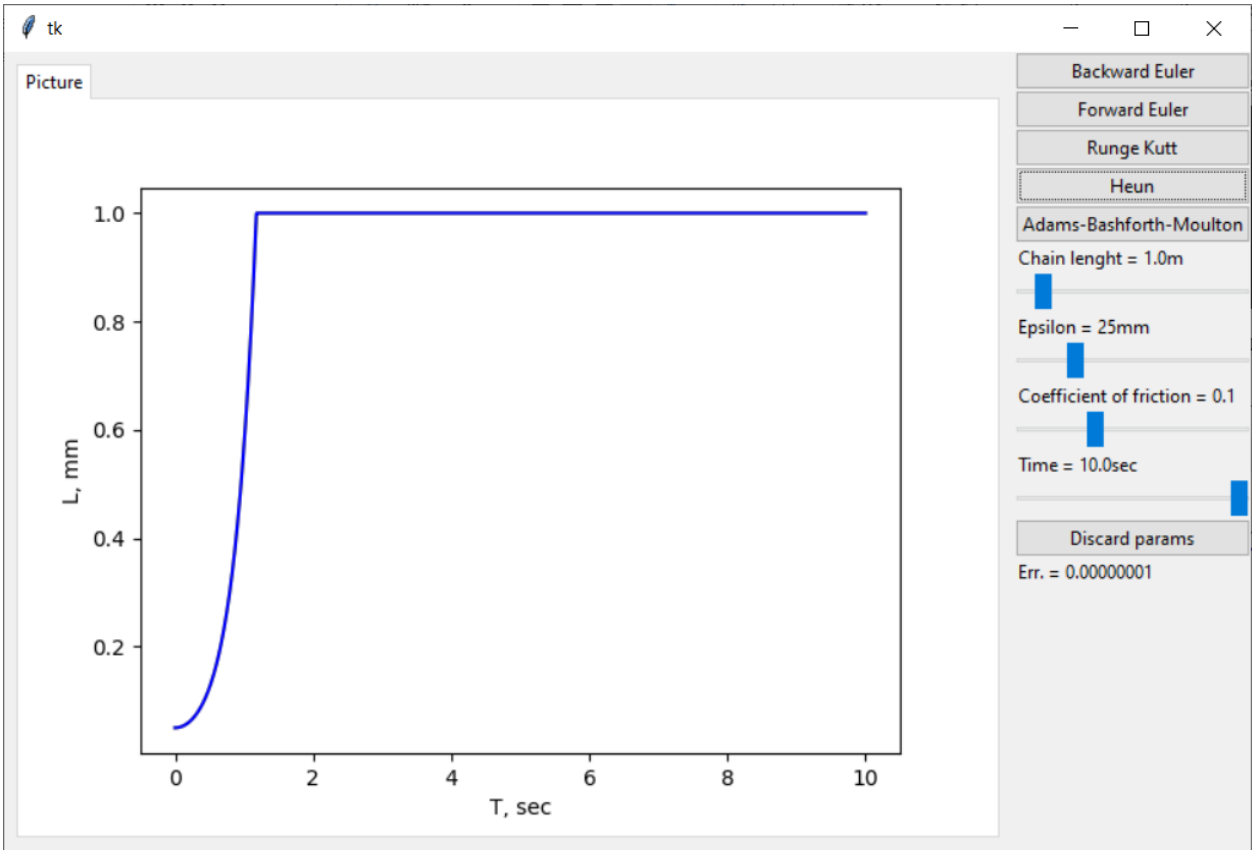
Figure 1
1e−15
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2000
4000
6000
8000
10000
12000

Метод Хойна

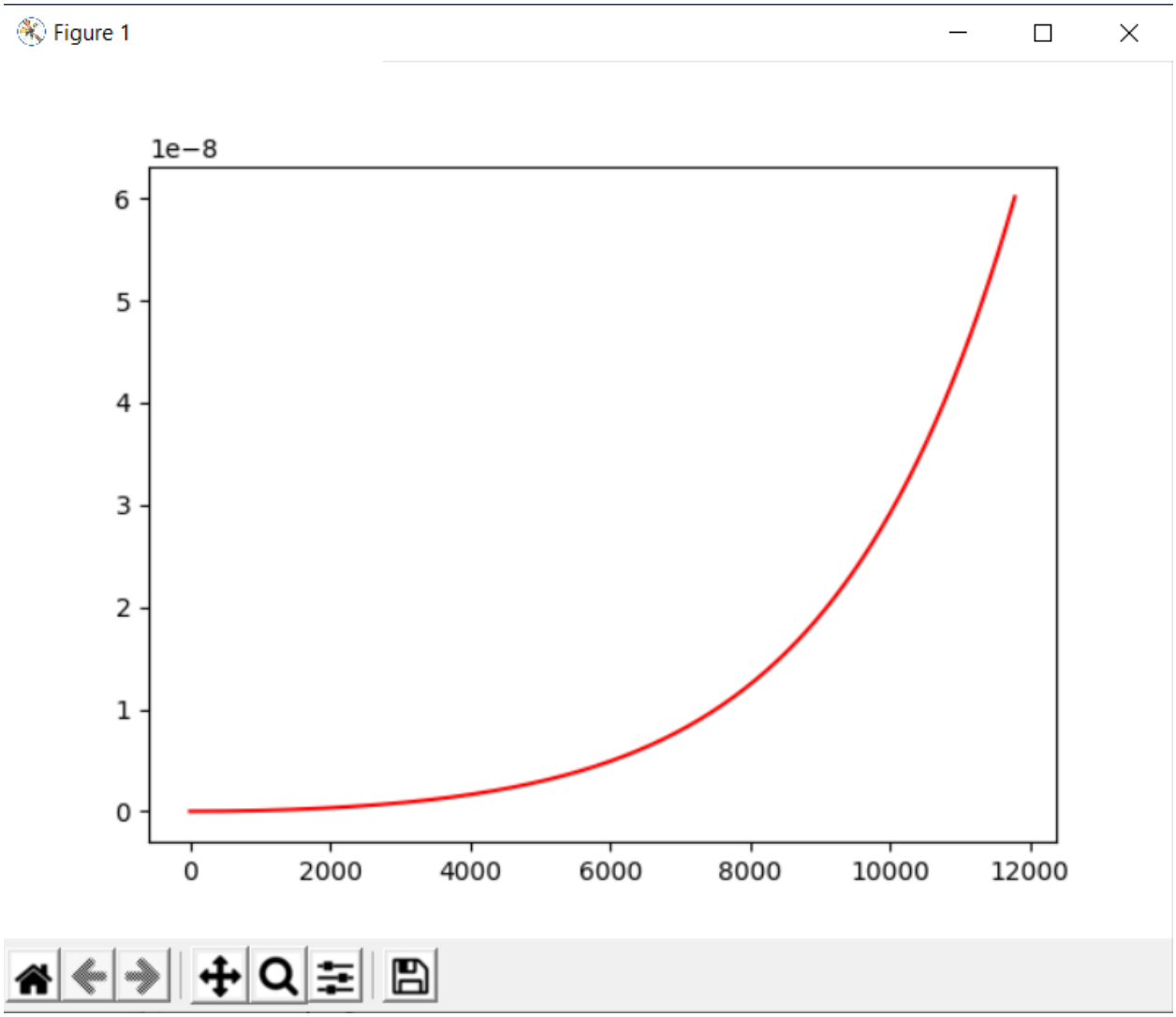
В работе метод был реализован так:

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

    v = 0
    y0 = self.coef * self.chain_len / (1 + self.coef) + 2 * self.eps/1000
    x = np.array([y0, v])
    res = []

    self.f.write(str(0) + ' ')

    for i in range(appr):

        xdot1 = np.array(mf.myFunc(x, self.coef, self.chain_len))
        xdot2 = np.array(mf.myFunc(x + xdot1*self.h, self.coef, self.chain_len))

        x = x + (xdot1 + xdot2) * self.h/2

        if (x[0] > self.chain_len):
            x[0] = self.chain_len
        res = np.append(res, x[0])
        self.f.write(str(i*self.h) + ' ')
    self.f.write('\n')

    return res
```

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.

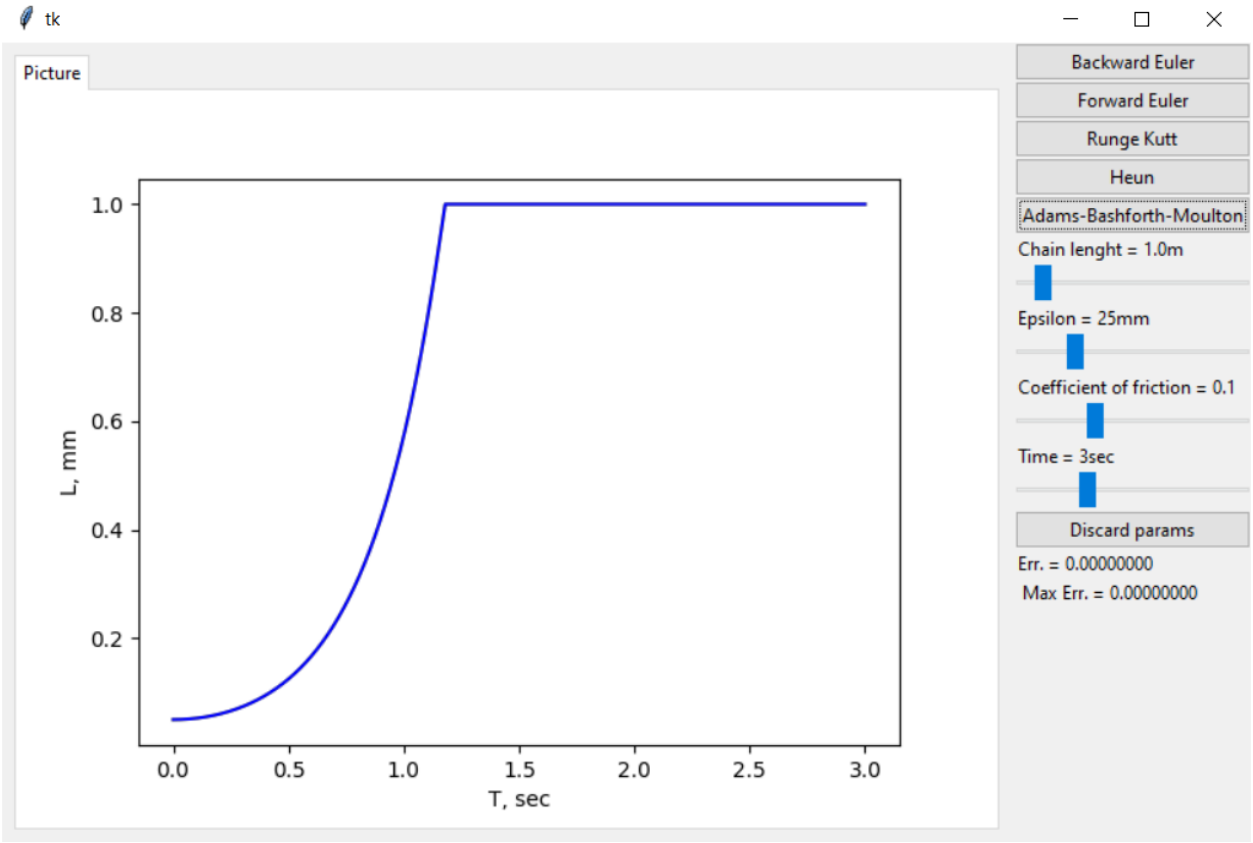


Метод Адамса-Башфорта 4-го порядка

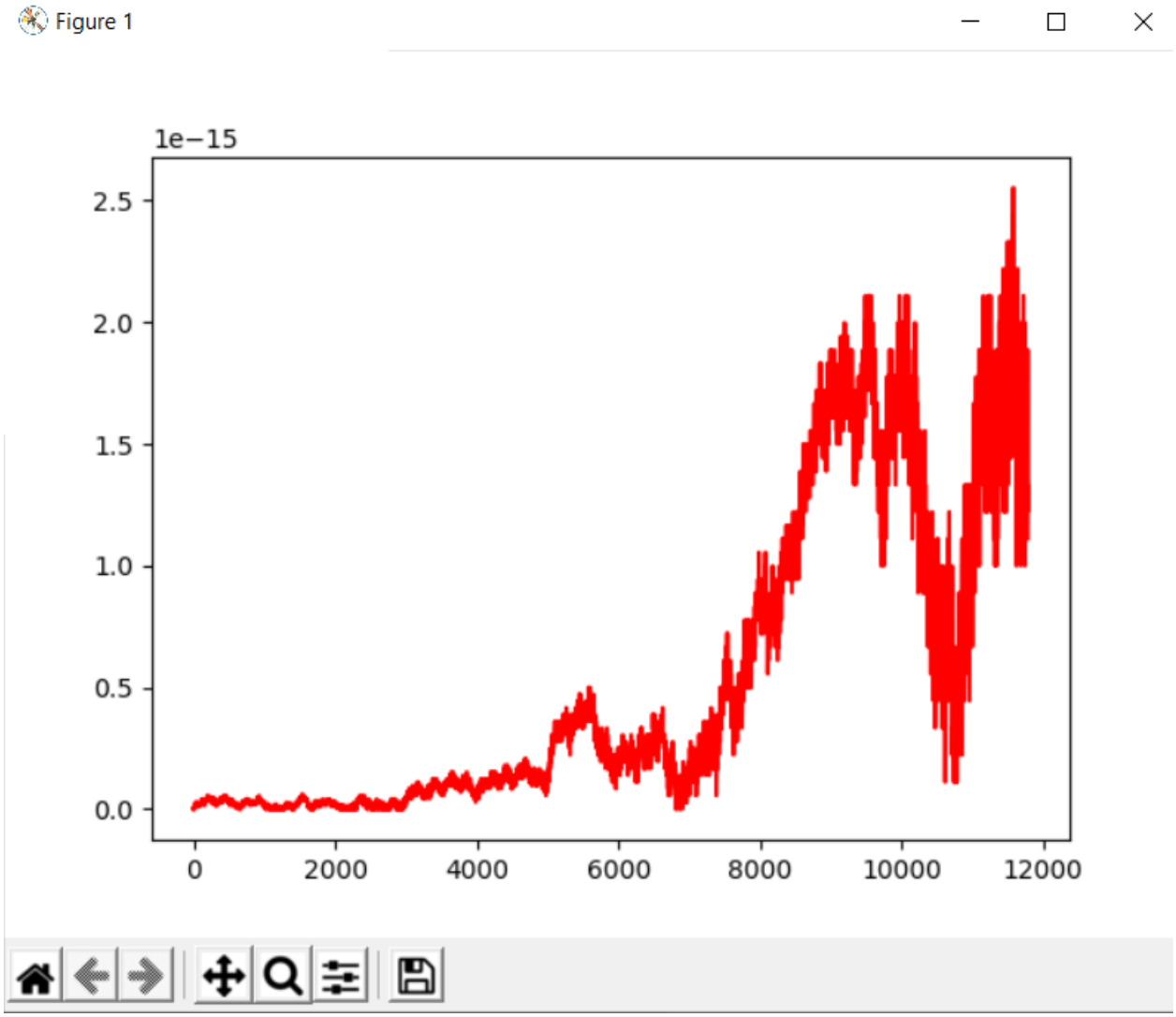
В работе метод был реализован так:

```
    t = 0

    for i in range(3, dx):
        xdot1 = np.array(mf.myFunc(x[i], self.coef, self.chain_len))
        xdot2 = np.array(mf.myFunc(x[i - 1], self.coef, self.chain_len))
        xdot3 = np.array(mf.myFunc(x[i - 2], self.coef, self.chain_len))
        xdot4 = np.array(mf.myFunc(x[i - 3], self.coef, self.chain_len))

        xp = x[i] + (self.h/24)*(55 * xdot1 - 59 * xdot2 + 37 * xdot3 - 9 * xdot4 )
        xdotp = np.array(mf.myFunc(xp, self.coef, self.chain_len))

        xn = x[i] + (self.h/24) * (9 * xdotp + 19 * xdot1 - 5 * xdot2 + xdot3)

        if (xn[0] > self.chain_len):
            xn[0] = self.chain_len
        # print (yn)

        t = t_temp[i] + self.h
        t_arr = np.append(t_arr, t)

        self.x = t_arr

        res = np.append(res, xn[0])
    return [t_arr, res]
```

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.



Сводная таблица методов

	Прямой метод Эйлера		Обратный метод Эйлера		Метод РКФ4-5		Метод Хойна		Метод Адамса Башфорта 4-го порядка	
h, N	h= 0.01, N = 300	h= 0.0001, N = 30000	h= 0.01, N = 300	h= 0.0001, N = 30000	h= 0.01, N = 300	h= 0.0001, N = 30000	h= 0.01, N = 300	h= 0.0001, N = 30000	h= 0.01, N = 300	h= 0.0001, N = 30000
Max. err	0.0500900956061466 34	0.0005357570531380196	0.0465736535915 9882	0.0005354085268496345	1.5097367800365 19e-11	1.6653345369377 348e-15	0.0005686965691512613	6.0135343260547 81e-08	7.1313583926979 62e-08	2.5535129566378 6e-15
Acc.	h= 0.000001 M = 3000000		h= 0.000001 M = 3000000		h= 0.01 M = 60		h= 0.001 M = 4		h= 0.03 M = 36	

h – размер шага, N - количество шагов, Max.Err – максимальная ошибка, Acc. – заданная точность, равная 1e-6, M – количество вызовов правой части, для достижения заданной точности с шагом h

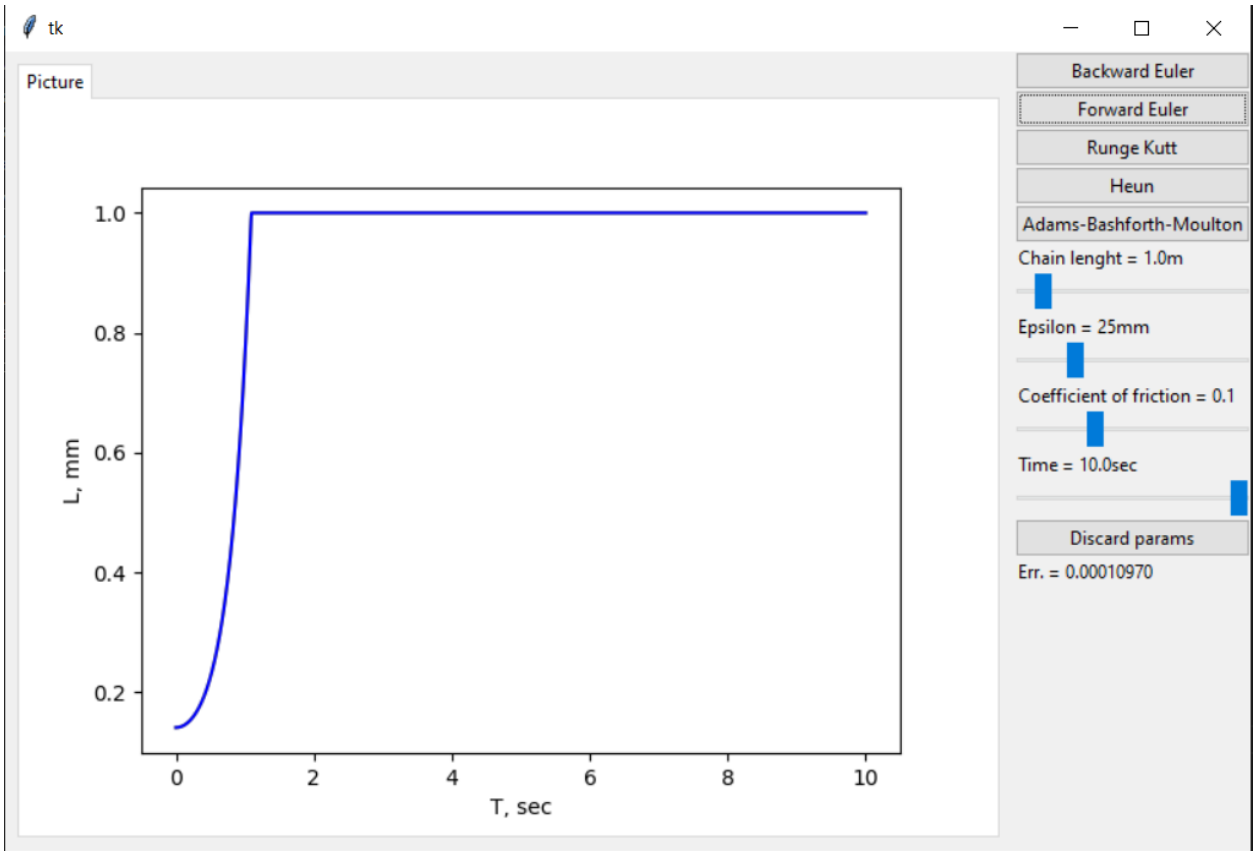
GUI

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

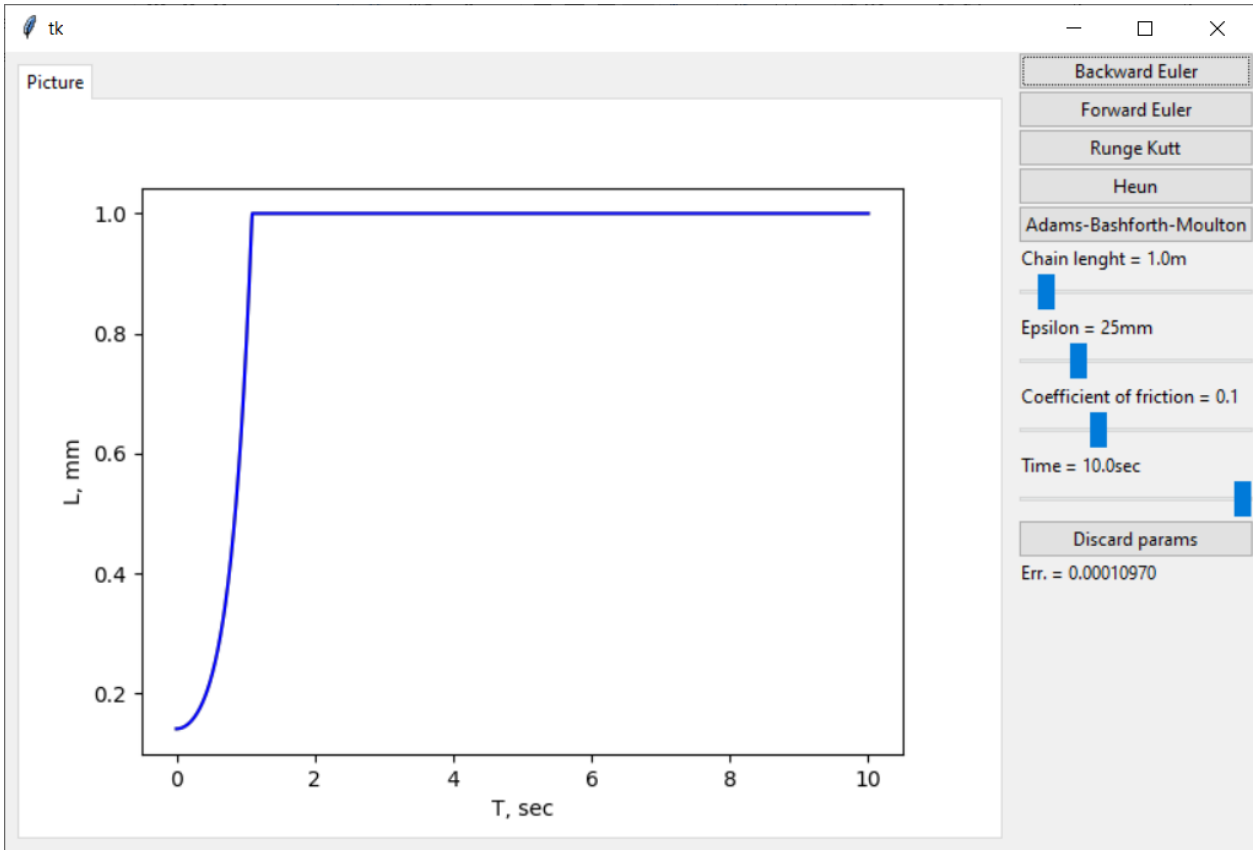
Интерфейс программы включает в себя 6 кнопок, 3 слайдера, окно для графиков и лэйбл с выводом информации об успешном/неуспешном запуске программы:

Первые пять кнопок в интерфейсе вызывают численные методы:

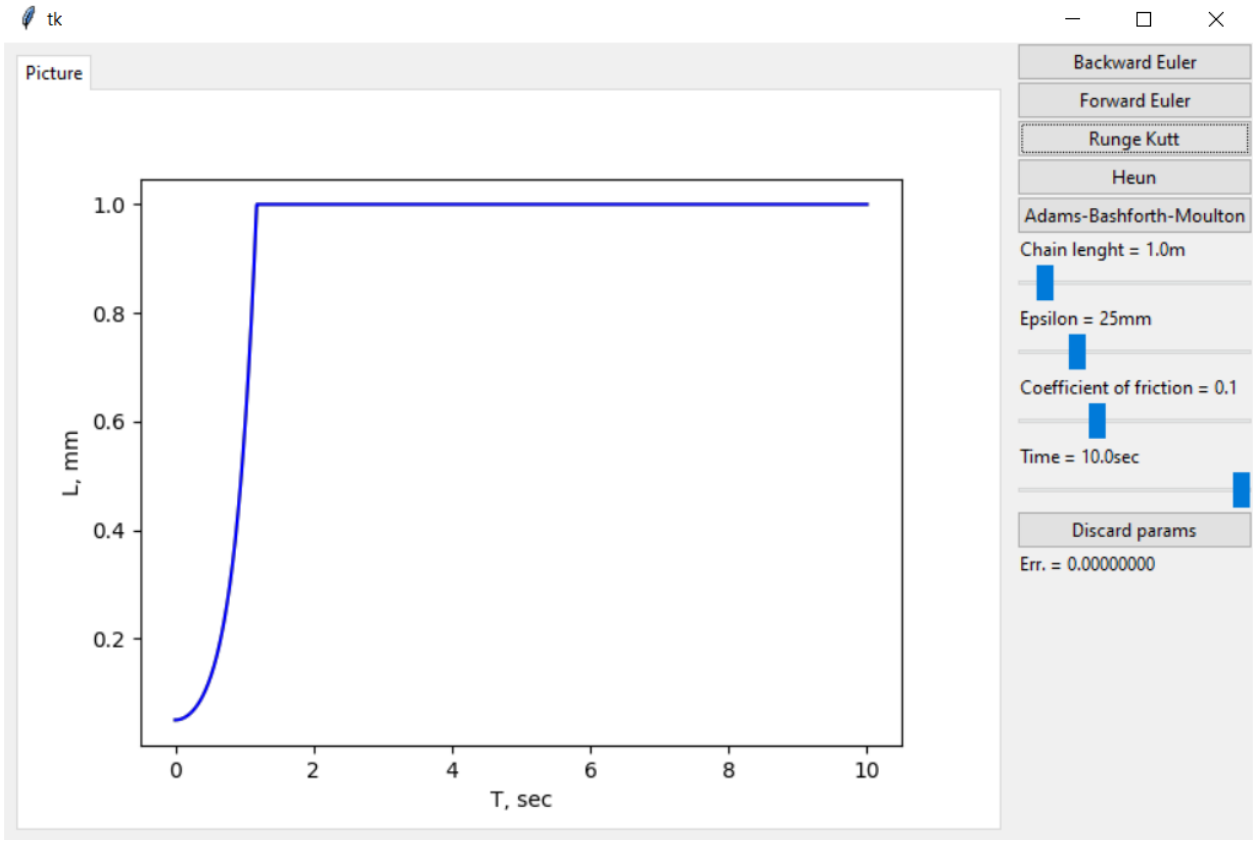
Forward Euler:



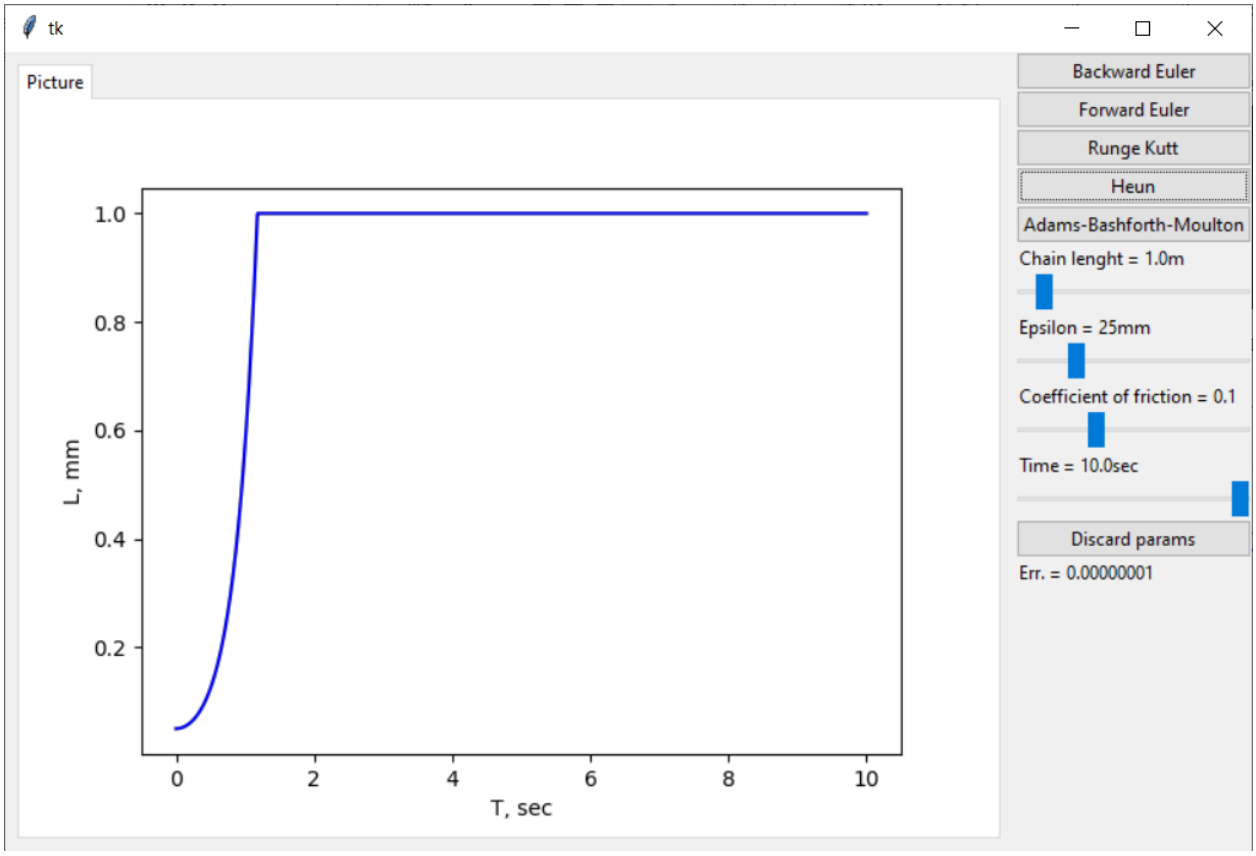
Backward Euler:



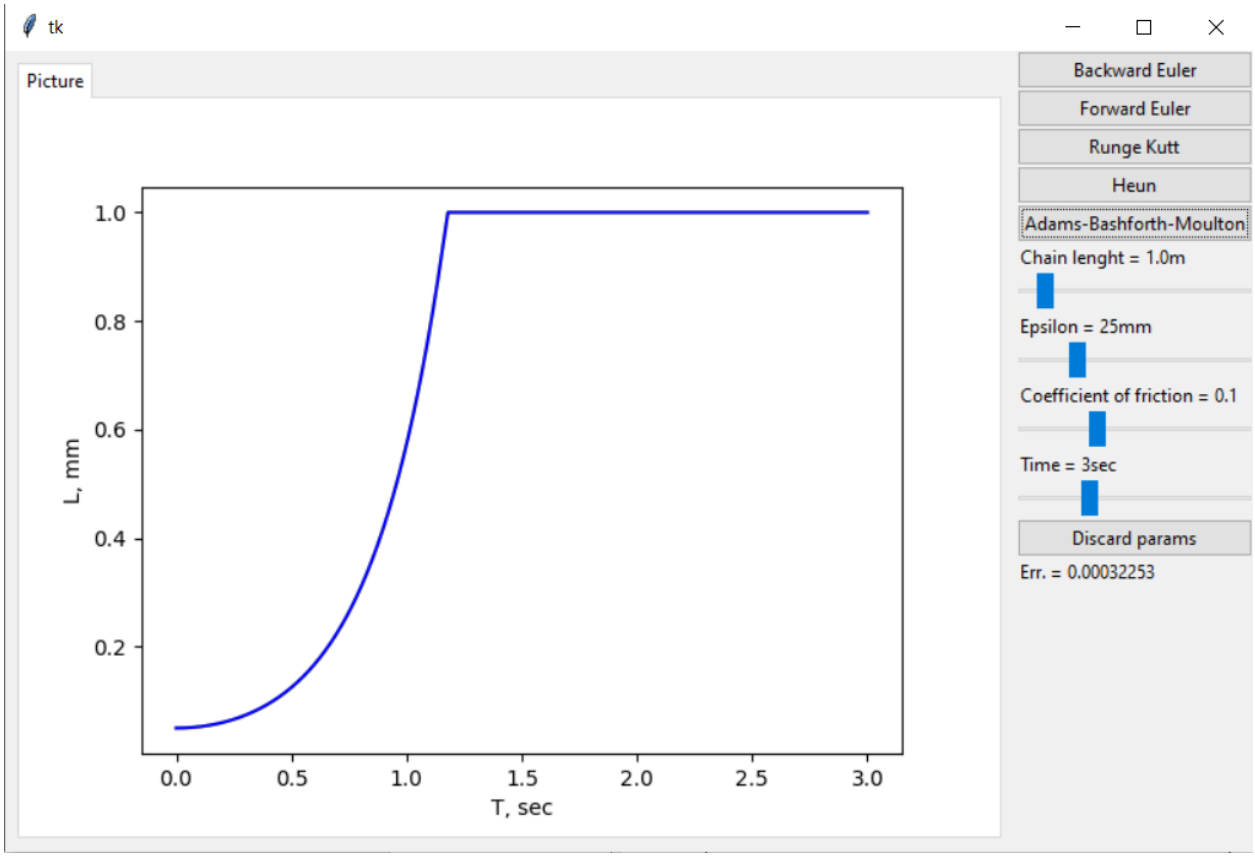
Runge Kutt:



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/