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

КУРСОВАЯ РАБОТА

по дисциплине «Дифференциальные уравнения»

Тема: Ракета

Гордиенко А.М.

Студенты гр. 8382 _____ Ершов М.И.

Преподаватель _____ Павлов Д.А.

Санкт-Петербург
2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Гордиенко А.М.
Студент Ершов М.И.
Группа 8382
Тема работы: Ракета

Исходные данные:
Ракета

Содержание пояснительной записки:
«Содержание», «Введение», «Прямой метод Эйлера», «Обратный метод Эйлера», «Метод Хойна», «Метод Рунге-Кутты 4 порядка», «Графический интерфейс», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 28.08.2021

Дата сдачи курсовой работы: 21.10.2021

Дата защиты курсовой работы: 21.10.2021

Студенты		Гордиенко А.М. Ершов М.И.
Преподаватель		Павлов Д.А.

АННОТАЦИЯ

В курсовой работе рассмотрена задача полета ракеты. Для этого использовалась формула Циолковского. Для решения поставленной задачи было использовано несколько методов: «Прямой метод Эйлера», «Обратный метод Эйлера», «Метод Хойна», «Метод Рунге-Кутты 4-го порядка». Результаты решения данного уравнения были представлены в виде графиков в графическом интерфейсе.

SUMMARY

In the course work, the problem of rocket flight. For this Tsiolkovsky rocket equation was used. To solve the problem, several methods were used: "Forward Newton's method", "Backward Newton's method", "Heun's method", "Runge-Kutta method of the 4^{th} 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. Полет ракеты является этим процессом. Были использованы методы интегрирования дифференциальных уравнений динамических систем такие как: «Прямой метод Эйлера», «Обратный метод Эйлера», «Метод Хойна», «Метод Рунге-Кутты 4-го порядка».

Выполнение работы

Была реализована программа, создающая графический интерфейс для ввода пользователем исходных данных ракеты, выбора численного метода, отрисовки графика зависимости скорости от времени.

Опишем начальные условия задачи в виде системы уравнений.

$$\frac{d}{dt}(mv) = F_{\text{Итог}}$$

$$\frac{dm}{dt}v + \frac{dv}{dt}m = R - mg - kv^2,$$

где m – масса ракеты, изменяющаяся со временем, v – скорость ракеты, R – постоянная тяга, g – гравитационная постоянная, k – постоянное сопротивление воздуха (была получена экспериментально), t – время.

Зная постоянную скорость расхода топлива λ, заменим массу на функцию массы от времени.

$$m(t) = m_0 - \lambda t.$$

Получаем

$$-\lambda v + (m_0 - \lambda t)\frac{dv}{dt} = R - (m_0 - \lambda t)g - kv^2.$$

Оставляем в одной части уравнения $\frac{dv}{dt}$:

$$\frac{dv}{dt} = \frac{R-(m_0-\lambda t)g-kv^2+\lambda v}{(m_0-\lambda t)}$$

Результат каждого численного метода сравнивался с эталонной функцией, которая имеет следующую формулу:

$$\Delta v = I_{sp} g_0 \ln\left(\frac{m_0}{m_f}\right),$$

Где I_{sp} – импульс, g_0 – гравитационная постоянная, m_0 – начальная масса, m_f – конечная масса (изменяемая во времени).

Каждый метод был наследован от абстрактного класса Processor, в котором инициализируются основные параметры ракеты, а также методы вычислений массы и скорости ракеты в данный момент времени.

Структура класса Processor представлена на рис. 1.

```
class Processor:
    def __init__(self, stages, thrust, mass, burn_rate, burn_time, h):
        self.g = 9.8
        self.stages = stages

        self.mass = mass
        self.thrust = [i * 1000 for i in thrust]  # 153.51 * 1000  # kN
        self.initial_mass = sum([i for i in mass])  # 3380  # kg
        self.burn_rate = burn_rate  # 87.37864  # kg / s
        self.burn_time = burn_time  # 10.3  # s

        self.specific_impulse = [self.thrust[i] / (self.g * self.burn_rate[i]) for i in range(self.stages)]
        self.drag_coefficient = 0.38  # kg / m

        self.final_mass = [self.initial_mass - self.burn_time[i] * self.burn_time[i] for i in range(stages)]
        self.h = h  # 0.5

    @staticmethod
    def m_t(mass, time, lamb):
        return mass - time*lamb

    def v_t(self, r, m, lam, t, g, k, v):
        return (r - g * self.m_t(m, t, lam) - k * v * v + lam * v) / self.m_t(m, t, lam)
```

Рисунок 1 – Структура класса Processor.

Прямой метод Эйлера

Метод представляет собой дискретное получение следующего значения путем приращения предыдущего значения на величину изменения функции, умноженной на шаг.

Алгоритм можно описать следующим образом:

$$v_{n+1} = v_n + hf(v_n, t_n),$$

где $f = \frac{dv}{dt}$ – ускорение, h - шаг.

Реализация метода представлена на рис. 2.

```python
import math
from processor import Processor


class ForwardEuler(Processor):
    def compute(self):
        sum_steps = 1
        v = [0]
        v_ = [0]
        mass = self.initial_mass
        for i in range(self.stages):
            steps = int(round(self.burn_time[i] / self.h, 0))
            for n in range(steps):
                v.append(v[-1] + self.h * self.v_t(self.thrust[i],
                                                   mass,
                                                   self.burn_rate[i], n, self.g,
                                                   self.drag_coefficient, v[-1]))
                print(f"Step: {sum_steps+n}, Xn: {sum_steps+n * self.h}, Vn: {v[-1]}")
                v_.append(self.specific_impulse[i] * self.g * math.log(mass / self.m_t(mass, n*self.h, self.burn_rate[i])))
            sum_steps += steps
            mass -= self.mass[i]
        return sum_steps, v, v_
```

Рисунок 2 – Реализация прямого метода Эйлера.

Обратный метод Эйлера

Обратный метод Эйлера схож с прямым методов.

$$v_{n+1} = v_n + hf(v_{n+1}, t_{n+1})$$

Реализация метода представлена на рис. 3:

```python
import math
from processor import Processor


class BackwardEuler(Processor):
    def compute(self):
        sum_steps = 1
        v = [0]
        v_ = [0]
        mass = self.initial_mass
        for i in range(self.stages):
            steps = int(round(self.burn_time[i] / self.h, 0))
            for n in range(steps):
                v.append(v[-1] + self.h * self.v_t(self.thrust[i],
                                                   mass,
                                                   self.burn_rate[i], n+1, self.g,
                                                   self.drag_coefficient, v[-1]))
                print(f"Step: {sum_steps+n}, Xn: {sum_steps+n * self.h}, Vn: {v[-1]}")
                v_.append(self.specific_impulse[i] * self.g * n * math.log(mass / self.m_t(mass, n, self.burn_rate[i])))
            sum_steps += steps
            mass -= self.mass[i]
        return sum_steps, v, v_
```

Рисунок 3 – Реализация обратного метода Эйлера.

Метод Хойна

Метод Хойна, или же трапецеидальный метод, можно интерпретировать как сочетание прямого и обратного методов Эйлера.

$$v_{n+1} = v_n + \frac{h}{2}(f(v_n, t_n) + f(v_{n+1}, t_{n+1}))$$

Реализация метода представлена на рис. 4:

```
import math
from processor import Processor


class Heun(Processor):
    def compute(self):
        sum_steps = 1
        v = [0]
        v_ = [0]
        mass = self.initial_mass
        for i in range(self.stages):
            steps = int(round(self.burn_time[i] / self.h, 0))
            for n in range(steps):
                v.append(v[-1] + (self.h / 2) * (
                    self.v_t(self.thrust[i],
                             mass,
                             self.burn_rate[i], n, self.g,
                             self.drag_coefficient, v[-1]) +
                    self.v_t(self.thrust[i],
                             mass,
                             self.burn_rate[i], n+1, self.g,
                             self.drag_coefficient, v[-1])))
                v_.append(self.specific_impulse[i] * self.g * math.log(mass / self.m_t(mass, n*self.h, self.burn_rate[i])))
                print(f"Step: {sum_steps+n}, Xn: {sum_steps+n * self.h}, Vn: {v[-1]}")
            sum_steps += steps
            mass -= self.mass[i]
        return sum_steps, v, v_
```

Рисунок 4 – Реализация метода Хойна.

Метод Рунге-Кутты 4-го порядка

$$v_{n+1} = v_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(v_n, t_n)$$

$$k_2 = f(v_n + \frac{h}{2}k_1, t_n + \frac{h}{2})$$

$$k_3 = f(v_n + \frac{h}{2}k_2, t_n + \frac{h}{2})$$

$$k_4 = f(v_n + \frac{h}{2}k_3, t_n + \frac{h}{2})$$

Реализация метода представлена на рис. 5.

```python
class RungeKutta(Processor):
    def compute(self):
        sum_steps = 1
        v = [0]
        v_ = [0]
        mass = self.initial_mass
        for i in range(self.stages):
            steps = int(round(self.burn_time[i] / self.h, 0))
            for n in range(steps):
                k1 = self.v_t(self.thrust[i],
                              mass,
                              self.burn_rate[i], n, self.g,
                              self.drag_coefficient, v[-1])
                k2 = self.v_t(self.thrust[i],
                              mass,
                              self.burn_rate[i], n + self.h/2, self.g,
                              self.drag_coefficient, v[-1] + k1 * self.h/2)
                k3 = self.v_t(self.thrust[i],
                              mass,
                              self.burn_rate[i], n + self.h/2, self.g,
                              self.drag_coefficient, v[-1] + k2 * self.h/2)
                k4 = self.v_t(self.thrust[i],
                              mass,
                              self.burn_rate[i], n + self.h/2, self.g,
                              self.drag_coefficient, v[-1] + k3 * self.h/2)
                v.append(v[-1] + (self.h / 6) * (k1 + 2 * k2 + 2 * k3 + k4))
                v_.append(self.specific_impulse[i] * self.g * n * math.log(mass / self.m_t(mass, n, self.burn_rate[i])))
                print(f"Step: {sum_steps+n}, Xn: {sum_steps+n-1 * self.h}, Vn: {v[-1]}")
            sum_steps += steps
            mass -= self.mass[i]
        return sum_steps, v, v_
```

Рисунок 5 – Реализация метода Хойна.

Сводная таблица методов

	Прямой метод Эйлера			Обратный метод Эйлера		
h, _	1	0.5	0.25	1	0.5	0.25
max_err	61.4641	56.2898	41.6607	57.4125	48.2738	46.4476
mean_err	25.0033	22.8877	22.81	14.2228	25.7076	13.6597
median_err	18.2794	18.1484	24.965	12.1596	27.7708	13.6911
mse_err	1024.686	853.3848	718.4678	370.617	922.4648	286.6761

Продолжение сводной таблицы методов

	Метод Хойна			Метод Рунге-Кутты 4-го порядка		
h, _	1	0.5	0.25	1	0.5	0.25
max_err	58.8753	44.963	51.95	2194.1812	12948.3911	68837.4210
mean_err	23.8825	24.2446	13.8624	661.1641	3999.2717	19824.7759
median_err	17.9295	26.3679	12.9149	357.7907	2588.121	12372.0313

mse_err	936.7793	817.1466	322.0555	965983.03	32220383.838	803204257.09

GUI

Графический интерфейс был написан на языке Python с использованием библиотеки trinter.

Интерфейс включает в себя поля ввода параметров ракеты, список методов, список с выбором шага алгоритма и кнопку запуска алгоритма.

Окно приложения представлено на рис. 6.

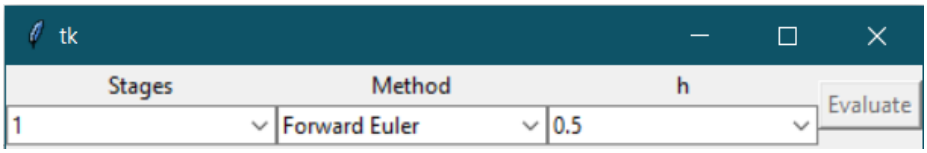


Рисунок 6 – Окно приложения в момент запуска.

После выбора пользователем количества ступеней появляются поля для ввода параметров ступеней.

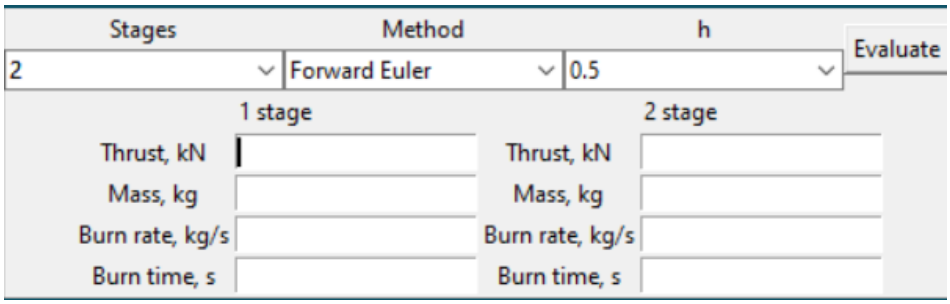


Рисунок 7 – Окно приложения с полями для ввода параметров ракеты.

Результаты работы методов представлены на рис. 8-10.

Forward Euler:

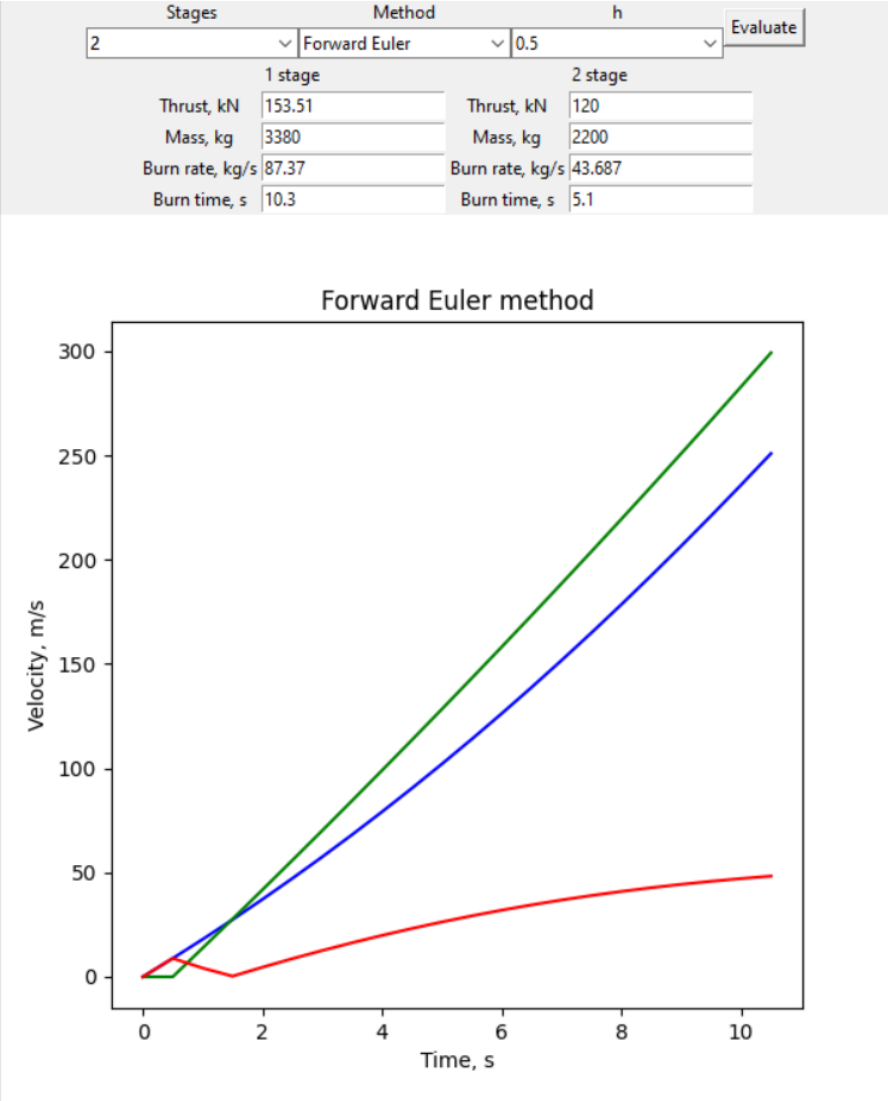
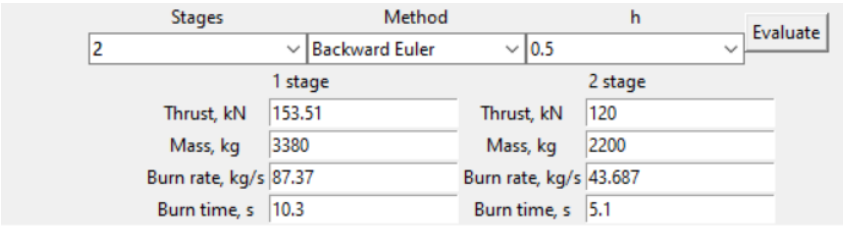


Рисунок 8 – Вывод программы прямого метода Эйлера.

Backward Euler:



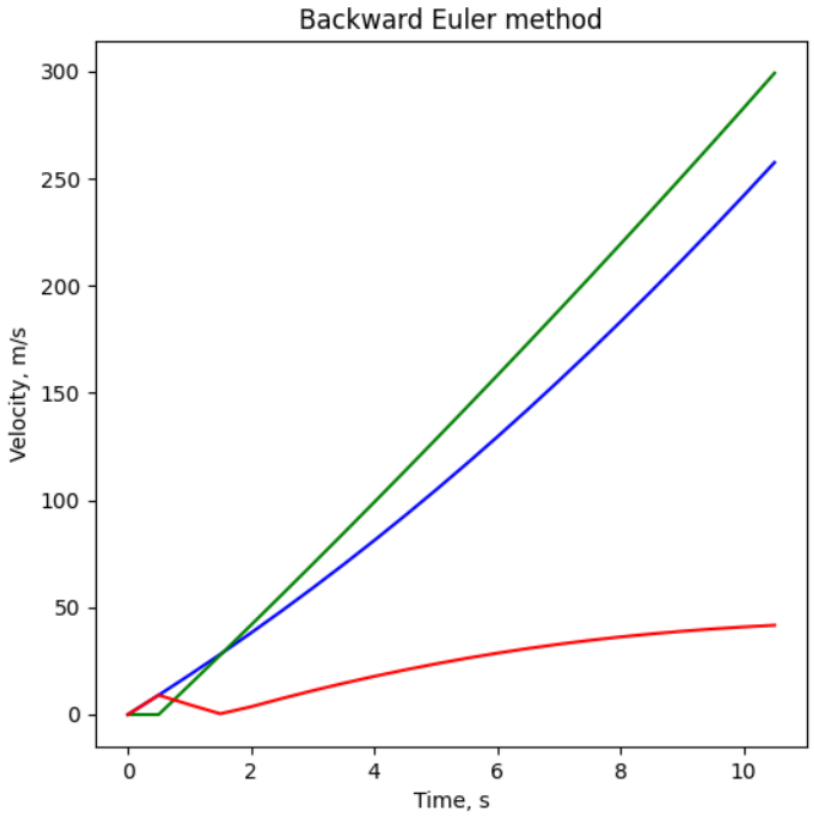


Рисунок 9 – Вывод программы обратного метода Эйлера.

Heun:

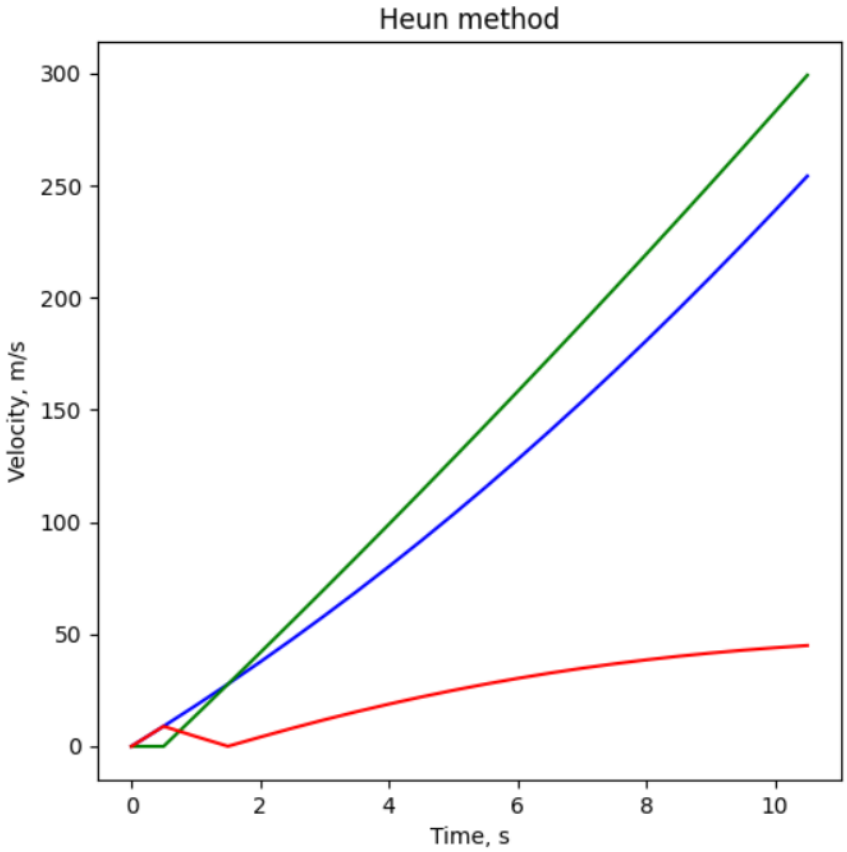


Рисунок 10 – Вывод программы метода Хойна.

С помощью выкидного списка выбирается количество ступеней ракеты, численный метод, шаг алгоритма.

В поля ввода каждой ступени пользователь вводит параметры соответствующей ступени.

После ввода пользователем нажимается кнопка Evaluate, которая выбранным пользователем методом проводит вычисления.

Результат работы метода выводится в виде графика в окне приложения.

На графике синей линией рисуется физический метод, зеленой – математический метод (эталонный), красной – значение ошибки.

Вывод

В ходе выполнения курсовой работы была написана программа, реализующая оконное приложение с численными методами, были изучены основные методы аппроксимации решения с непрерывным дискретным временем.

Используемая литература

https://www.python.org/

http://chaos.sgu.ru/K52/MND/algoritms/algoritms.html

https://www.simiode.org/resources/8310/download/SIMIODE_EXPO_2021_B1-R2_Christopher_Scott_Vaughen.pdf

https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%A6%D0%B8%D0%BE%D0%BB%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%BE%D0%B3%D0%BE

https://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%9C%D0%B5%D1%89%D0%B5%D1%80%D1%81%D0%BA%D0%BE%D0%B3%D0%BE