Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт
Высшая школа прикладной математики и вычислительной физики

КУРСОВАЯ РАБОТА

Решение задачи Коши при помощи численных методов

по дисциплине

«Практикум по языкам программирования»

Работу выполнил:
Г. Е. Родин
Группа 5030301/20002

Преподаватель:
А.А. Смирновский

“____” _____2023 г.

Санкт-Петербург
2023

Оглавление

Введение

В настоящей курсовой работе решается задача Коши, имеющая физический смысл.

Отталкиваясь от условия задачи, записано дифференциальное уравнение (сформулирована задача Коши). Решение обычного дифференциального уравнения (далее ОДУ) получено при помощи языка программирования Python.

В работе приведено аналитическое решение дифференциального уравнения при помощи встроенной библиотеки SymPy. А также два численных решения методом Эйлера и Рунге-Кутты 4-го порядка.

Графики полученных функций – решений ОДУ представлены при помощи встроенной библиотеки MatPlotLib.

Проанализирована зависимость погрешностей численных методов от шага и определен порядок точности обоих методов при помощи методов логарифмической невязки и экстраполяции по Ричардсону.

Исследовано поведение графика функции при различных входных параметрах.

1. Формулировка исследуемой задачи

1.1. Условие задачи

Частица движется замедленно по окружности радиуса r так, что ее тангенциальное и нормальное ускорения в каждый момент времени равны друг другу по модулю. В начальный момент точке была сообщена скорость v0.

Найти скорость точки как функцию пройденного пути s.

1.2. Формулировка задачи Коши

$$a_n = \frac{v^2}{R} - \text{нормальное ускорение,}$$

$$a_t = \frac{dv}{dt} - \text{тангенцальное ускорение}$$

$$a_n = a_t \leftrightarrow \frac{dv}{dt} = -\frac{v^2}{R}$$

Получили ОДУ вида $f(v,t) = \frac{dv}{dt}$, причем $v(t=0) = v_0$, знак минус говорит о том, что тело движется, замедляясь.

Далее ещё понадобится зависимость пройденного пути от времени:

$$\frac{dS}{dt} = v, \text{причем } S(t=0) = 0$$

2. Аналитическое решение, полученное самостоятельно

2.1. Решение дифференциальных уравнений

Зависимость скорости от времени

$$\frac{dv}{dt} = -\frac{v^2}{R} \leftrightarrow \frac{dv}{v^2} = -Rdt \leftrightarrow -\frac{1}{R}\int_0^t dt = \int_{v_0}^{v} \frac{dv}{v^2} \leftrightarrow$$

$$\leftrightarrow -\frac{1}{R}(t-0) = \left(-\frac{1}{v} - \left(-\frac{1}{v_0}\right)\right) \leftrightarrow t = R\left(\frac{1}{v} - \frac{1}{v_0}\right)$$

$$v = \frac{1}{\frac{t}{R} + \frac{1}{v_0}}, \qquad t = R\left(\frac{1}{v} - \frac{1}{v_0}\right)$$

Зависимость расстояния от времени

$$\frac{dS}{dt} = v \leftrightarrow dS = \frac{dt}{\frac{t}{R} + \frac{1}{v_0}} \leftrightarrow \int_0^S dS = \int_0^t \frac{dt}{\frac{t}{R} + \frac{1}{v_0}} \leftrightarrow$$

$$\leftrightarrow (S-0) = R\int_0^t \frac{d(\frac{t}{R} + \frac{1}{v_0})}{\frac{t}{R} + \frac{1}{v_0}} \leftrightarrow S = R(\ln\left(\frac{t}{R} + \frac{1}{v_0}\right) - \ln\left(\frac{0}{R} + \frac{1}{v_0}\right)) \leftrightarrow$$

$$S = R(\ln\left(\frac{t}{R} + \frac{1}{v_0}\right) + \ln(v_0)) \leftrightarrow S = Rln(\frac{tv_0}{R} + 1)$$

$$S = Rln(\frac{tv_0}{R} + 1)$$

Зависимость скорости от пути

$$S = Rln\left(\frac{tv_0}{R} + 1\right), \qquad t = R\left(\frac{1}{v} - \frac{1}{v_0}\right) \leftrightarrow S = Rln\left(\frac{v_0}{v} - \frac{v_0}{v_0} + 1\right) \leftrightarrow$$

$$S = Rln\left(\frac{v0}{v}\right) \leftrightarrow e^{\frac{S}{R}} = \frac{v0}{v} \leftrightarrow v = v_0 e^{-\frac{S}{R}}$$

$$v = v_0 e^{-\frac{S}{R}}$$

2.2. График полученной зависимости

Для построения графика было использовано приложение GeoGebra, для примера было взято значение начальной скорости $v_0 = 20\frac{\text{м}}{\text{с}}$ и радиус окружности $R = 5$ м.

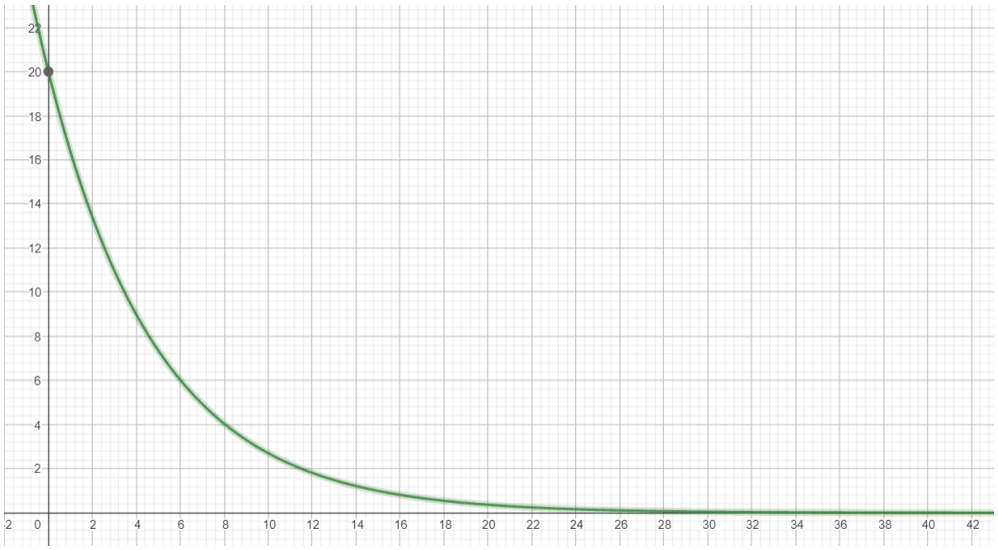


Рис. 1 График полученной зависимости в GeoGebra

$$f: y = 20\, e^{-\frac{x}{5}}$$

Рис. 2 Функция, чей график был построен

3. Аналитическое решение, полученное на Python

3.1. Решение дифференциальных уравнений

```
import sympy as sym

#Создание символьных перменных
t, v0, r= sym.symbols('t v0 r')
v, s = sym.symbols('v, s', cls = sym.Function)

#Первое дифференциальное уравнение для скорости
eq1 = sym.Eq(v(t).diff(t), -v(t)**2/r)
v1 = sym.dsolve(eq1,ics={v(0):v0})

#Зависимость скорости от времени
v1
```

$$v(t) = -\frac{r}{-\frac{r}{v_0} - t}$$

Листинг 1 Зависимость скорости от времени

Заметим, что $v(t) = -\frac{r}{-\frac{r}{v_0}-t} = \frac{r}{\frac{r}{v_0}+t} = \frac{1}{\frac{t}{r}+\frac{1}{v_0}}$, результат совпадает с самостоятельным решением.

```
#Второе дифференциальное уравнение для пройденного пути
eq2 = sym.Eq(s(t).diff(t), 1/(1/v0+t/r))
s1 = sym.dsolve(eq2, ics={s(0):0})

#Зависимость Пройденного пути от времени
s1
```

$$s(t) = -r\log(r) + r\log(r + tv_0)$$

Листинг 2 Зависимость пройденного пути от времени

Заметим, что $s(t) = -rln(r) + rln(r + tv_0) = r\left(\ln\left(\frac{r+tv_0}{r}\right)\right) =$

$= rln(1 + \frac{tv_0}{r})$, результат также совпадает с самостоятельным решением.

```
#Задаем связь скорости и времени
v = sym.symbols('v')
exp = Eq(v - 1/(1/v0 + t/r))

#Выражаем время через скорость
sym.solve(exp, t)
```

```
[-r/v0 + r/v]
```

Листинг 3 Зависимость времени от скорости

Получили $t = -\frac{r}{v_0} + \frac{r}{v} = r(\frac{1}{v} - \frac{1}{v_0})$

```
#Задаем зависимость S и v
v = sym.symbols('v')
expr = r*sym.ln(1 + t*v0/r)
expr = expr.subs(t, r/v-r/v0)

#Находим как зависит S от v
expr1 = sym.simplify(expr1)

#Находим зависимость v от S
s = sym.symbols('s')
exp1 = sym.Eq(s, expr1)
sym.solve(exp1, v)
```

```
[v0*exp(-s/r)]
```

Листинг 4 Связь v и S

Получили $v = v_0 e^{-\frac{s}{r}}$, результат совпал с решением, полученным самостоятельно.

3.2. График полученной зависимости

Получили массив значений полученной функции и представили его в виде графика.

```
import matplotlib.pyplot as plt
import numpy as np
#Создание массивов для графиков
x = np.linspace(0, 40, 100)
y = 20*np.exp(-x/5)

#Построение графика
plt.plot(x, y, color='orange')
plt.title('Зависимость скорости от пройденного пути')
plt.xlabel("S")            # подпись оси абсцисс
plt.ylabel("v")
plt.grid()
plt.show()
```

Листинг 5 Код для построения графика

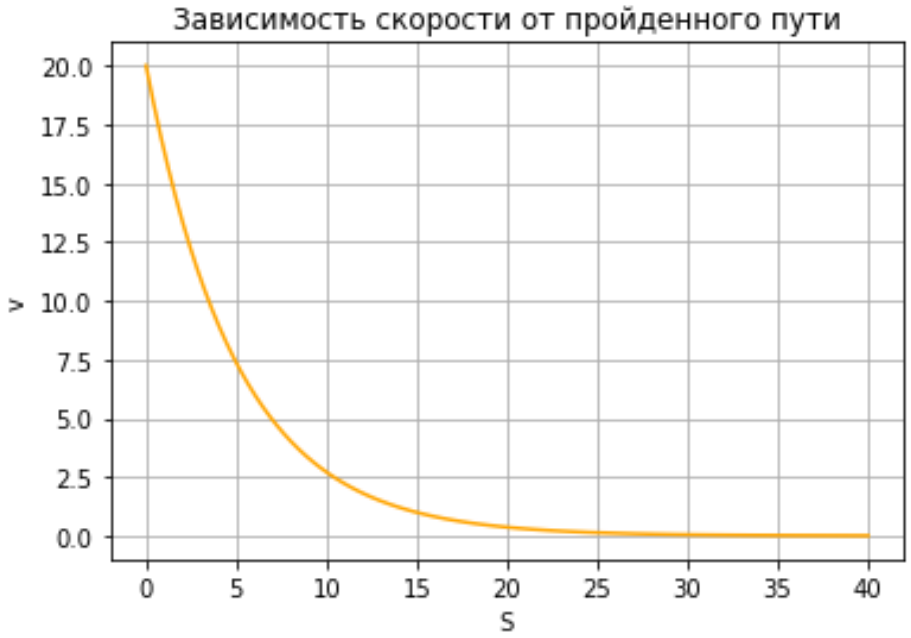


Рис. 3 Полученный график

Как можно видеть, график, полученный при помощи библиотеки MatPlotlib такой же, какой мы получили в приложении GeoGebra.

4. Реализация численных методов

Реализуем на языке Python два численных метода, метод Эйлера и метод Рунге-Кутты 4-ого порядка, входные параметры будем вводить через файл, а в результате выполнения программы получим 2 текстовых файла с массивами результатов выполнения методов и графики полученных зависимостей.

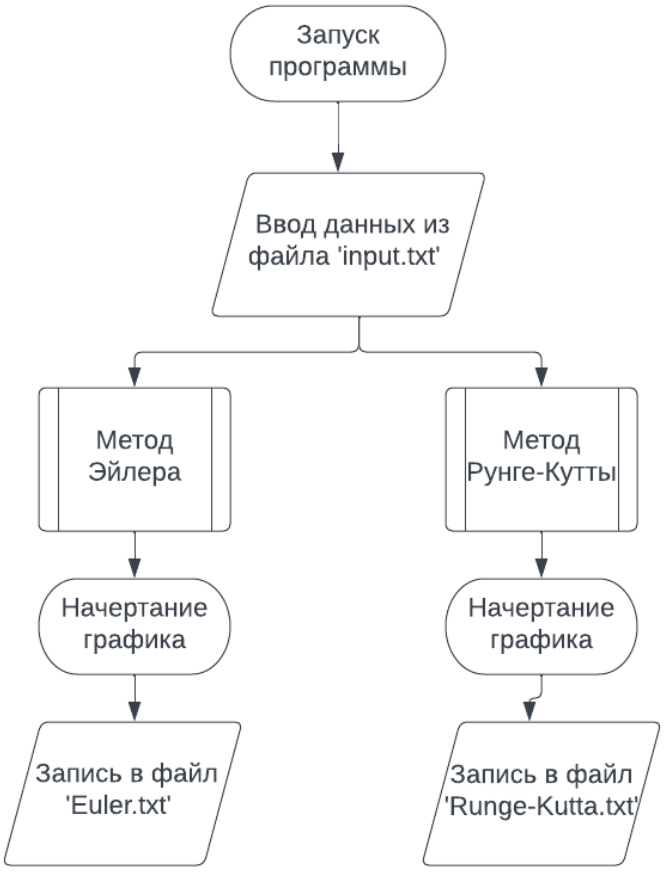


Рис. 4 Блок-схема кода для численных методов

4.1. Метод Эйлера

4.1.1. Описание численного метода

Метод Эйлера – простейший численный метод решения ОДУ.

Пусть дана задача Коши для уравнения первого порядка:

$$\frac{dy}{dx} = f(x, y), \qquad y(x = x_0) = y_0$$

Решение ищется на интервале $[x_0, b]$, на этом промежутке введём узлы $x_0 < x_1 < \cdots < x_n < b$. Приближенное решение в узлах x_i, которое обозначим через y_i, определяется по формуле $y_i = y_{i-1} + h \cdot f(x_{i-1}, y_{i-1})$,

$$h = (x_i - x_{i-1})$$

4.1.2. Реализация на Python

```
#Функция для дифференциального уравнения
def f(v, t, r):
    return -v**2 / r

#Метод Эйлера
def euler_method(h, v0, r):
    t = [0]
    v = [v0]
    s = [0]

    while True:
        v_next = v[-1] + h * f(v[-1], t[-1], r) #Считаем значение скорости
        t_next = t[-1] + h
        s_next = s[-1] + v[-1] * h #Считаем значение пройденного пути

        #Добавляем найденные значения в точке в соответствующие массивы
        v.append(v_next)
        t.append(t_next)
        s.append(s_next)

        if abs(v[-1] - v[-2]) < 0.0001 * h: #Условие выхода, разность двух послледних
                break # значений скоростей меньше одной десятитысячной шага

    return s, v
```

Листинг 6 Код Метода Эйлера

```
#Считываем параметры из файла
with open("input.txt", "r") as file:
    params = file.read().splitlines()

h = float(params[0])
v0 = float(params[1])
r = float(params[2])
```

Рис. 5 Считывание параметров из файла

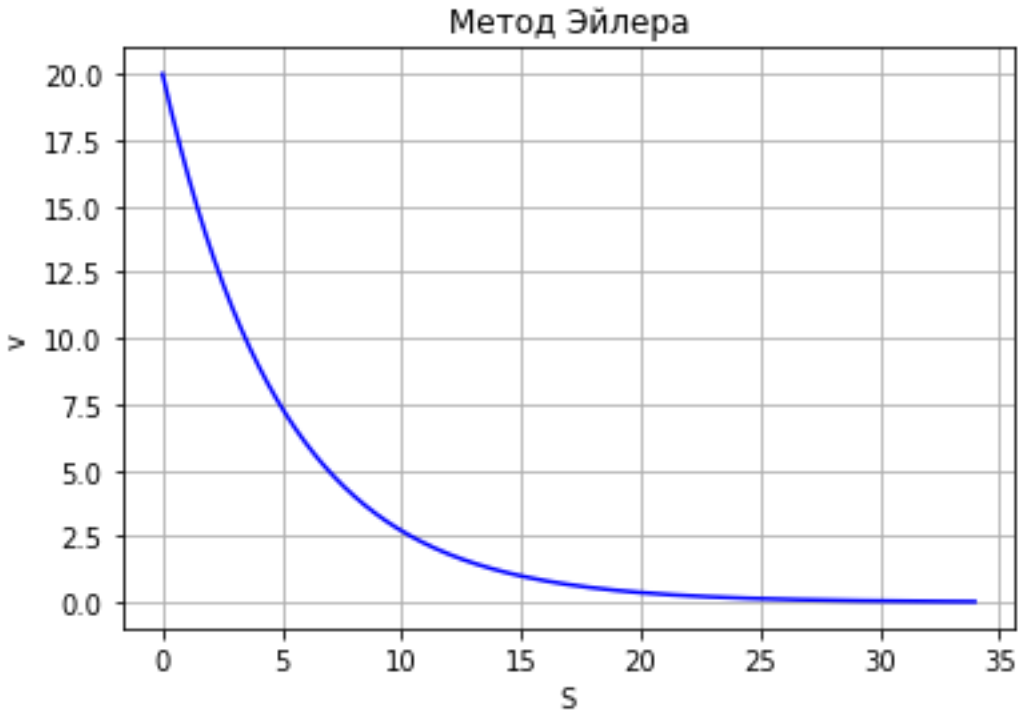


Рис. 6 График для метода Эйлера

В результате выполнения кода был получен график численного решения дифференциального уравнения, визуально они похожи, степень их схожести оценим в дальнейших пунктах.

4.2. Метод Рунге-Кутты 4-ого порядка

4.2.1. Описание численного метода

Метод Рунге-Кутты – высокоточный численный метод решения ОДУ. Он является продолжением, усовершенствованной версией метода Эйлера.

Первая часть описания аналогична методу Эйлера, отличие заключается в следующем:

$$\mathbf{k}_1 = \mathbf{f}(x_n, \mathbf{y}_n),$$
$$\mathbf{k}_2 = \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right),$$

приближенное значение в последующих точках вычисляется по итерационной формуле: $y_i = y_{i-1} + h \cdot (k_1 + 2k_2 + 2k_3 + k_4)$, то есть вычисление следующего значения идет в 4 этапа:

$$\mathbf{k}_3 = \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right),$$
$$\mathbf{k}_4 = \mathbf{f}(x_n + h, \mathbf{y}_n + h\,\mathbf{k}_3).$$

где h — величина шага сетки по x.

4.2.2. Реализация на Python

```
def runge_kutta_method(h, v0, r):
    t = [0]
    v = [v0]
    s = [0]

    while True:
        # Реализация алгоритма метода для скорости
        k1 = h * f(v[-1], t[-1], r)
        k2 = h * f(v[-1] + 0.5 * k1, t[-1] + 0.5 * h, r)
        k3 = h * f(v[-1] + 0.5 * k2, t[-1] + 0.5 * h, r)
        k4 = h * f(v[-1] + k3, t[-1] + h, r)
        v_next = v[-1] + (k1 + 2 * k2 + 2 * k3 + k4) / 6 #Считаем значение скорости
        t_next = t[-1] + h

        #Реализация алгоритма метода для пройденного пути
        c1 = h * f1(s[-1], t[-1], r, v0)
        c2 = h * f1(s[-1], t[-1] + 0.5 * h, r, v0)
        c3 = h * f1(s[-1], t[-1] + 0.5 * h, r, v0)
        c4 = h * f1(s[-1], t[-1] + h, r, v0)
        s_next = s[-1] + (c1 + 2 * c2 + 2 * c3 + c4) / 6

        v.append(v_next)
        t.append(t_next)
        s.append(s_next)

        if abs(v[-1] - v[-2]) < 0.001 * h: #Условие выхода как и в методе Эйлера
                break
    return s, v
```

Листинг 7 Код для метода Р-К-4

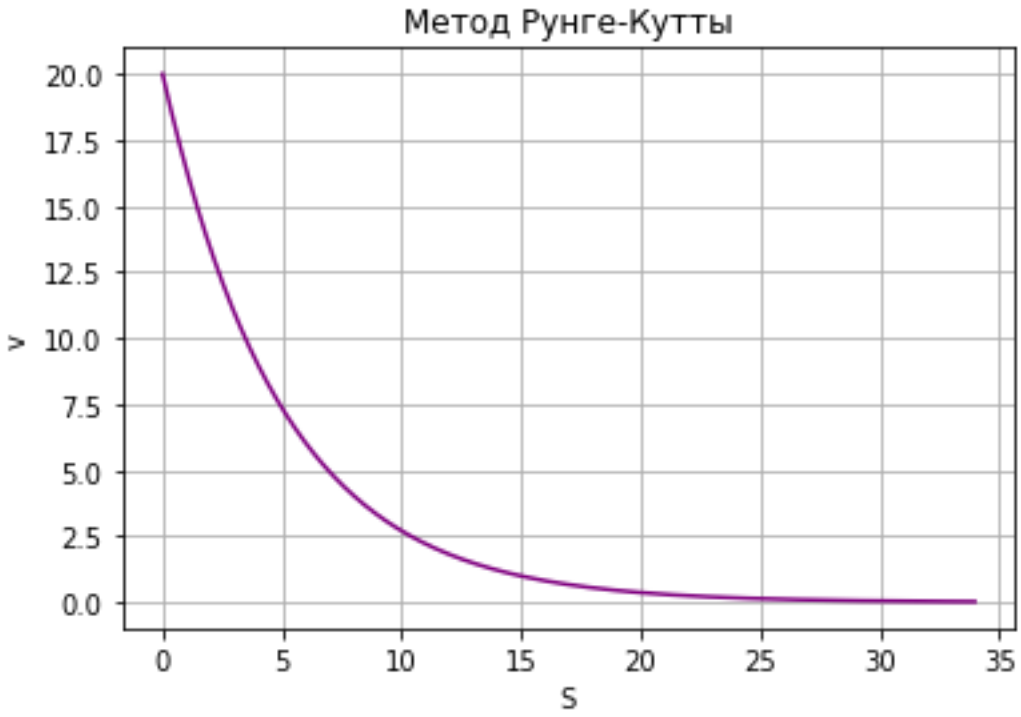


Рис. 7 График для метода Р-К-4

В результате выполнения кода снова был получен график численного решения дифференциального уравнения, визуально они похожи, степень их схожести также оценим уже в следующем пункте.

5. Погрешность численных методов

5.1. Общая сходимость

Посмотрим в целом как выглядят три полученных графика относительно друг друга, посмотрим, как они выглядят при различных значениях шага численного метода:

Рис. 8 Графики методов и теории (h=0.1)

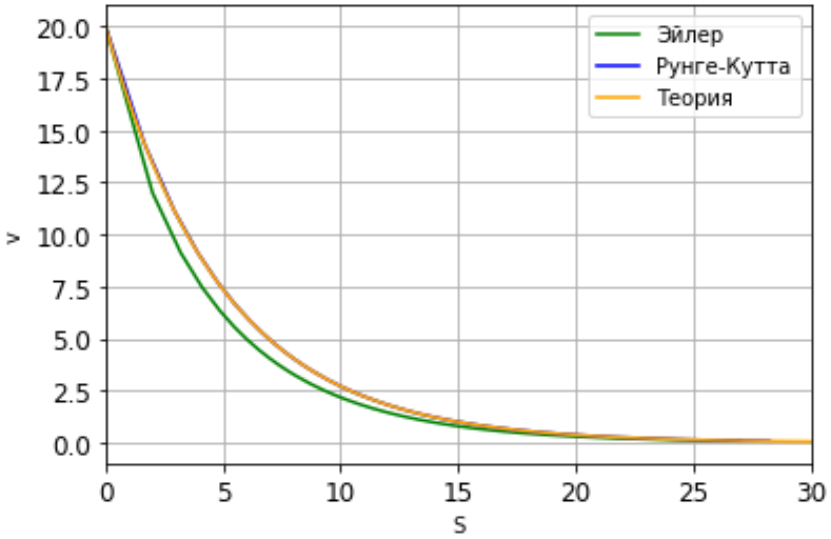


Рис. 9 Графики методов и теории (h=0.02)

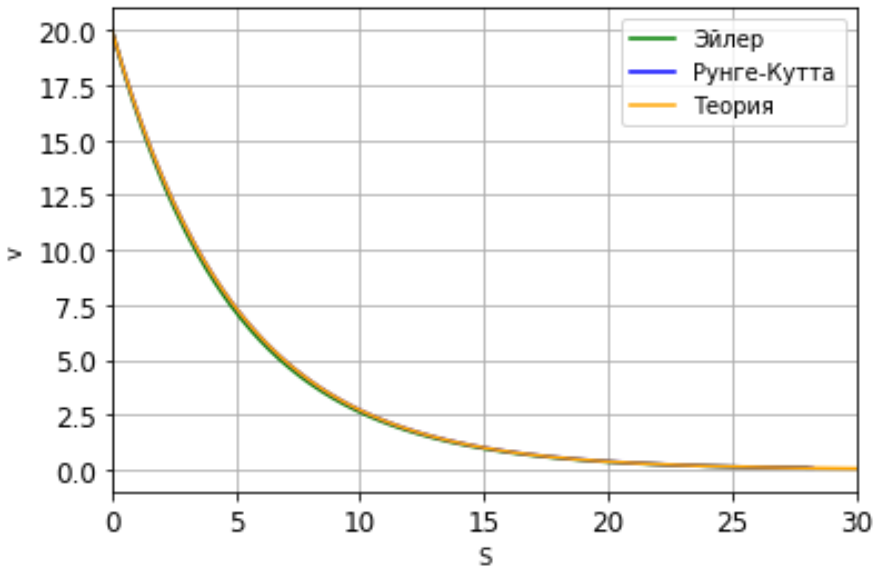
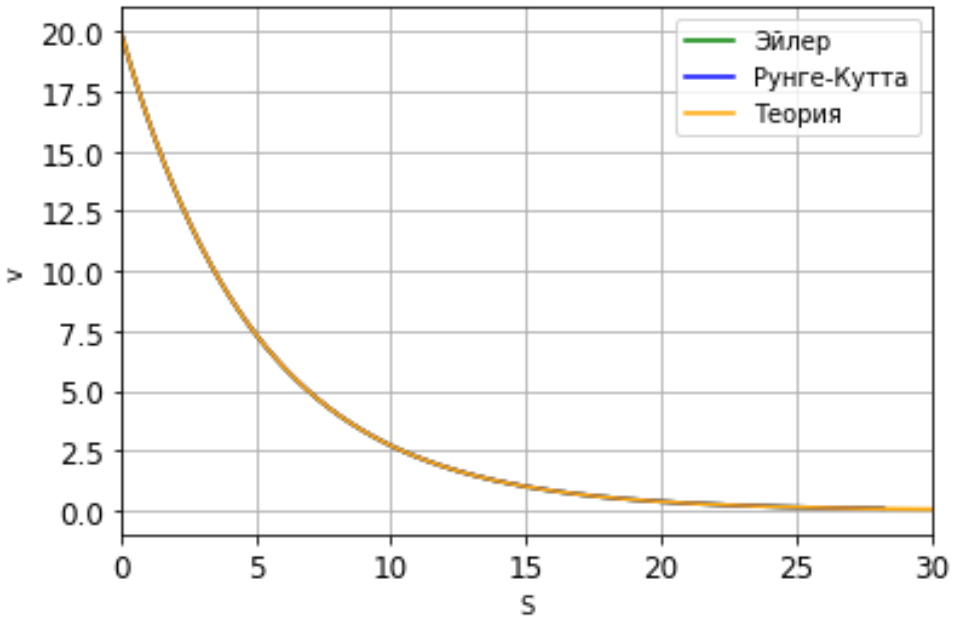


Рис. 10 Графики методов и теории (h=0.001)



Как можно заметить, для шага h=0.001 методы достаточно хорошо сходятся с теорией, а метод Рунге-Кутты уже и для шага h=0.1 практически полностью совпадает с теорией. Но теперь это нужно определить численно.

5.2. Исследование сходимости

Рассмотрим в цикле средние погрешности методов при шагах h=0.1, 0.01, 0.001, 0.0001 ($h = 10^{-i}$). Определение средней погрешности численного метода будет дано дальше.

Рис. 11 Блок схема для подсчета погрешностей

Абсолютная погрешность в точке считается по формуле:

$$err_{\text{абс}}(S_i) = \left|v(S_i) - v_{\text{теор}}(S_i)\right|$$

Относительная в точке считается по формуле

$$err_{\text{отн}}(S_i) = \frac{\left|v(S_i) - v_{\text{теор}}(S_i)\right|}{v_{\text{теор}}(S_i)}$$

Средняя погрешность метода считается по формуле:

$$\frac{\sum_{i=0}^{N} err_{\text{отн}}(S_i)}{N} = \frac{1}{N}\sum_{i=0}^{N}\frac{\left|v(S_i) - v_{\text{теор}}(S_i)\right|}{v_{\text{теор}}(S_i)}$$

Где N – число точек численного решения.

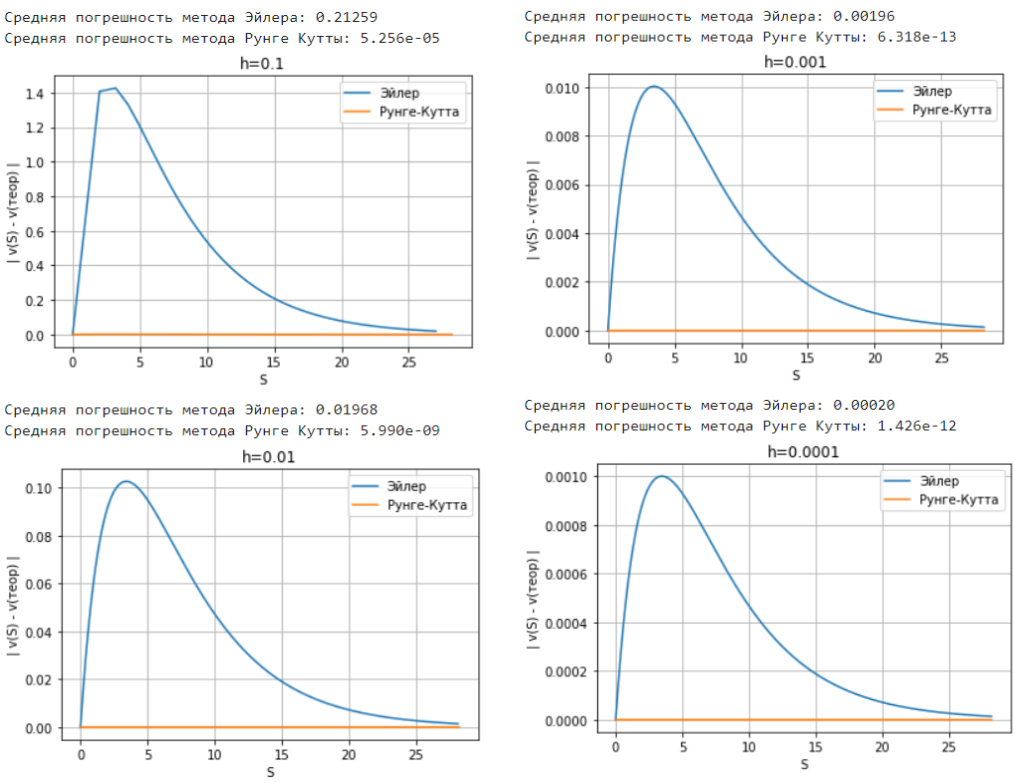


Рис. 12 Зависимость модуля погрешности от S для разных h

```
#Зависимость ошибки от шага
osh1 = []
osh2 = []
for i in range(1, 5):
    h1 = 10**(-i)
    (s1, v1) = euler_method(h1, v0, r)
    (s2, v2) = runge_kutta_method(h1, v0, r)

    #Погрешность для Метода Эйлера
    v_sum = 0
    eul_pogr = []
    for i in range (len(v1)):
        v_temp = abs(v1[i]-v0*np.exp(-s1[i]/r))
        v_sum += v_temp/(v0*np.exp(-s1[i]/r))
        eul_pogr.append(v_temp)
    v_sum /= len(v1)
    osh1.append(v_sum)
    print(f"Средняя погрешность метода Эйлера: {v_sum:.5f}")

    #Погрешность для Метода Рунге-Кутты
    v_sum = 0
    r_k_pogr = []
    for i in range (len(v2)):
        v_temp = (abs(v2[i]-v0*np.exp(-s2[i]/r)))
        v_sum += v_temp/(v0*np.exp(-s2[i]/r))
        r_k_pogr.append(v_temp)
    v_sum /= len(v2)
    osh2.append(v_sum)
    print(f"Средняя погрешность метода Рунге Кутты: {v_sum:.3e}")

    plt.plot(s1, eul_pogr, s2, r_k_pogr)
    plt.grid()
    plt.xlabel('S')
    plt.ylabel('| v(S) - v(теор) |')
    plt.title('h='+str(h1))
    plt.legend(["Эйлер", "Рунге-Кутта"])
    plt.show()
```

Листинг 8 Код для зависимости погрешности от S

Как можно видеть из графиков погрешность с уменьшением шага стремится к нулю.

Средняя погрешность метода Эйлера меньше 0.1% (1/1000 часть) для h=0.001, для метода Рунге-Кутты 4-ого порядка уже для h=0.1.

Таблица зависимости погрешности численного метода от величины шага

h	Метод Эйлера	Метод Рунге-Кутты
0.1	0.21259	5.256e-05
0.01	0.01968	5.990e-09
0.001	0.00196	6.318e-13
0.0001	0.00020	1.426e-12

Из таблицы можно заметить, что погрешность метода Рунге-Кутты возросла при уменьшении шага от 0.001 до 0.0001, это связано с тем, что погрешность стала меньше машинного нуля и программа перестала выдавать реальные значения, а лишь пороговые.

5.3. Метод логарифмических невязок

Посчитаем логарифм по основанию десять для средних погрешностей для четырех взятых шагов для обоих методов. Построим график зависимости логарифма средней ошибки от логарифма шага для двух методов, по тангенсу угла наклона графиков можно будет определить порядок точности методов.

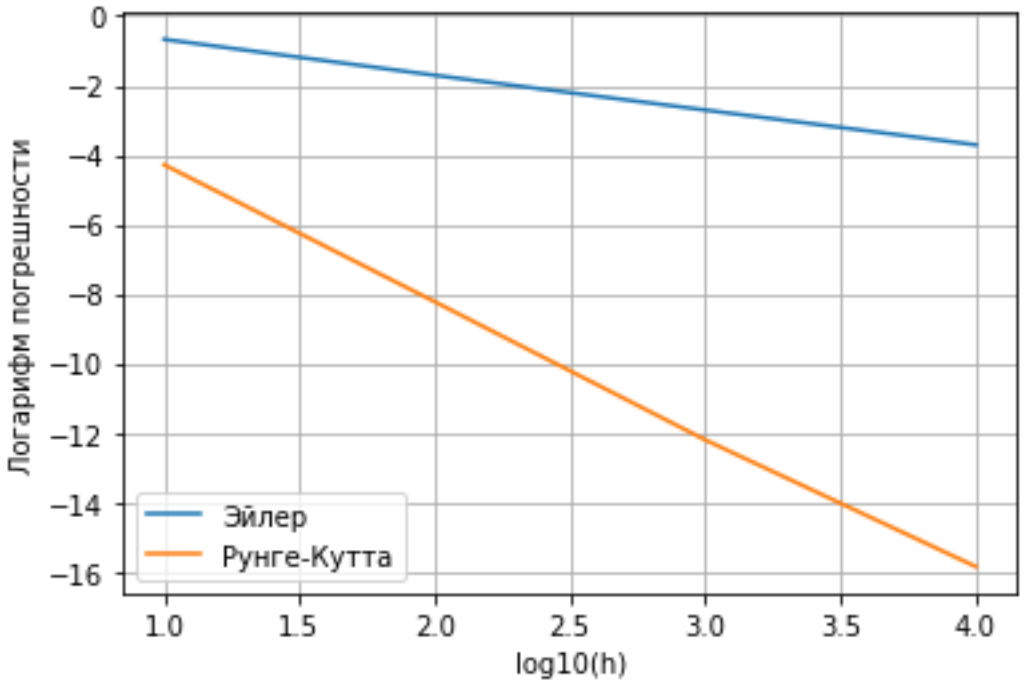


Рис. 13 График логарифмической зависимости погрешности от шага

```
#Код не компилируется с числами типа np.float128, поэтому погрешность для h=0.0001 Метода Рунге-Кутты на несколько порядков больше реальной
#я заменю osh2[3] на 7.326e-17 (osh2[3]/2 * 0.0001)
osh2[3] =  7.326*10**(-17)
Osh1 = [np.log10(osh1[i]) for i in range(4)]
Osh2 = [np.log10(osh2[i]) for i in range(4)]
arg = range(1, 5)

# Графики логарифмических зависимостей погрешности численных методов от шага
plt.plot(arg, Osh1, arg, Osh2)
plt.grid()
plt.ylabel('Логарифм погрешности')
plt.xlabel('log10(h)')
plt.legend(["Эйлер", "Рунге-Кутта"])

#Метод логарифмических невязок
Oh_Eu, Oh_RK = 0, 0
for i in range(3):
    Oh_Eu += Osh1[i] - Osh1[i + 1]
    Oh_RK += Osh2[i] - Osh2[i + 1]
Oh_Eu /= 3
Oh_RK /= 3
print('O(h) для метода Эйлера:', Oh_Eu)
print('O(h) для метода Рунге-Кутты:', Oh_RK)
```

```
O(h) для метода Эйлера: 1.0117590410890465
O(h) для метода Рунге-Кутты: 3.951938618857671
```

Листинг 9 Метод логарифмических невязок на Python

В итоге мы получили, что порядок точности метода Эйлера $O(h)$, а метода Рунге-Кутты 4-ого порядка $O(h^4)$, чего и следовало ожидать.

5.4. Экстраполяция по Ричардсону

Необходимо вычислить решение с тремя разными шагами: h, h/2, h/4. Тогда порядок точности метода p может быть оценён по формуле:

$p = log_2(\frac{y(h)-y(h/2)}{y(h/2)-y(4)})$, где y(h), y(h/2), y(h/4) – значения функции при соответствующих значениях шага.

Определим погрешность метода Эйлера

```
#Экстраполяция Ричардсона

#Погрешность метода Эйлера
(sh4, vh4) = euler_method(0.004, v0, r)
(sh2, vh2) = euler_method(0.002, v0, r)
(sh1, vh1) = euler_method(0.001, v0, r)

#Определяем значение скорости в момент, когда тело прошло 5 м
with open("Euler1.txt", "w") as file:
    for i in range(len(sh1)):
        file.write(f"S = {sh1[i]:.3f}, v = {vh1[i]:.3f}\n")

with open("Euler2.txt", "w") as file:
    for i in range(len(sh2)):
        file.write(f"S = {sh2[i]:.3f}, v = {vh2[i]:.3f}\n")

with open("Euler4.txt", "w") as file:
    for i in range(len(sh4)):
        file.write(f"S = {sh4[i]:.3f}, v = {vh4[i]:.3f}\n")

#Для v(h/4) взяли среднее между 7.331 и 7.288
p = np.log(abs(7.288-7.331)/abs(7.331-7.310))/np.log(2)
print(p)
```

Листинг 9 Реализация Экстраполяции Ричардсона для метода Эйлера

Определим значения функций скоростей для h=0.004, 0.002 и 0.001 соответственно в точке S~5 из соответствующих сгенерированных файлов.

```
S = 4.960, v = 7.407
S = 4.967, v = 7.396
S = 4.975, v = 7.386
S = 4.982, v = 7.375
S = 4.989, v = 7.364
S = 4.997, v = 7.353
S = 5.004, v = 7.342
S = 5.012, v = 7.331
S = 5.019, v = 7.321
S = 5.026, v = 7.310
S = 5.033, v = 7.299
```

```
S = 4.931, v = 7.441
S = 4.946, v = 7.419
S = 4.961, v = 7.397
S = 4.976, v = 7.375
S = 4.991, v = 7.353
S = 5.005, v = 7.331
S = 5.020, v = 7.310
S = 5.035, v = 7.288
S = 5.049, v = 7.267
S = 5.064, v = 7.246
S = 5.078, v = 7.225
```

```
S = 4.843, v = 7.554
S = 4.874, v = 7.508
S = 4.904, v = 7.463
S = 4.933, v = 7.419
S = 4.963, v = 7.375
S = 4.993, v = 7.331
S = 5.022, v = 7.288
S = 5.051, v = 7.246
S = 5.080, v = 7.204
S = 5.109, v = 7.162
```

Рис. 13 Значения функции из соответсвующих файлов

```
#Погрешность метода Рунге-Кутты
(sh4, vh4) = runge_kutta_method(0.04, v0, r)
(sh2, vh2) = runge_kutta_method(0.02, v0, r)
(sh1, vh1) = runge_kutta_method(0.01, v0, r)

#Определяем значение скорости в момент, когда тело прошло 10 м
with open("RungK1.txt", "w") as file:
    for i in range(len(sh1)):
        file.write(f"S = {sh1[i]:.3f}, v = {vh1[i]:.8f}\n")

with open("RungK2.txt", "w") as file:
    for i in range(len(sh2)):
        file.write(f"S = {sh2[i]:.3f}, v = {vh2[i]:.8f}\n")

with open("RungK4.txt", "w") as file:
    for i in range(len(sh4)):
        file.write(f"S = {sh4[i]:.3f}, v = {vh4[i]:.8f}\n")

p = np.log(abs(2.70270585 - 2.70270291)/abs(2.70270291 - 2.70270272))/np.log(2)
print(p)
```

Листинг 10 Реализация Экстраполяции Ричардсона для метода Рунге-Кутта

Определим значения функций скоростей для h=0.04, 0.02 и 0.01 соответственно в точке S~10.

S = 9.672, v = 2.89017701	S = 9.843, v = 2.79329631	S = 9.926, v = 2.74725276
S = 9.786, v = 2.82486219	S = 9.898, v = 2.76243115	S = 9.953, v = 2.73224045
S = 9.898, v = 2.76243423	S = 9.953, v = 2.73224065	S = 9.980, v = 2.71739132
S = 10.007, v = 2.70270585	S = 10.007, v = 2.70270291	S = 10.007, v = 2.70270272
S = 10.114, v = 2.64550566	S = 10.061, v = 2.67379699	S = 10.034, v = 2.68817206
S = 10.219, v = 2.59067647	S = 10.114, v = 2.64550284	S = 10.061, v = 2.67379680
S = 10.322, v = 2.53807384	S = 10.167, v = 2.61780124	S = 10.088, v = 2.65957448

Рис. 14 Значения функции из соответсвующих файлов

Получили порядок точности для двух методов, результаты совпадают с предыдущими:

Метод Эйлера: p = 1.03, метод Рунге-Кутты 4-ого порядка: p = 3.95.

По итогу двух проверок на точность численных методов мы получили: точность метода Эйлера $O(h^4)$, а метода Рунге-Кутты 4-ого порядка $O(h^4)$.

6. Исследовани функции при различных входных параметрах

Ввиду большей эффективности, будем использовать для исследования поведения функции при различных входных параметрах метод Рунге-Кутты, возьмем шаг h=0.01.

6.1. Исследование при различных значениях радиуса окружности

Сперва будем исследовать как себя ведет график при различных значениях радиуса окружности, по которой двигается тело. Рассмотрим график при значениях радиуса R=range(20, 3, -2).

```
s = []
v = []
l = []
for R in range(20, 3 ,-2):
    s_t, v_t = runge_kutta_method(0.01, v0, R)
    plt.plot(s_t, v_t)
    lab = 'R='+str(R)
    l.append(lab)

plt.legend(l)
plt.grid()
```

Рис. 26 Код для исследования

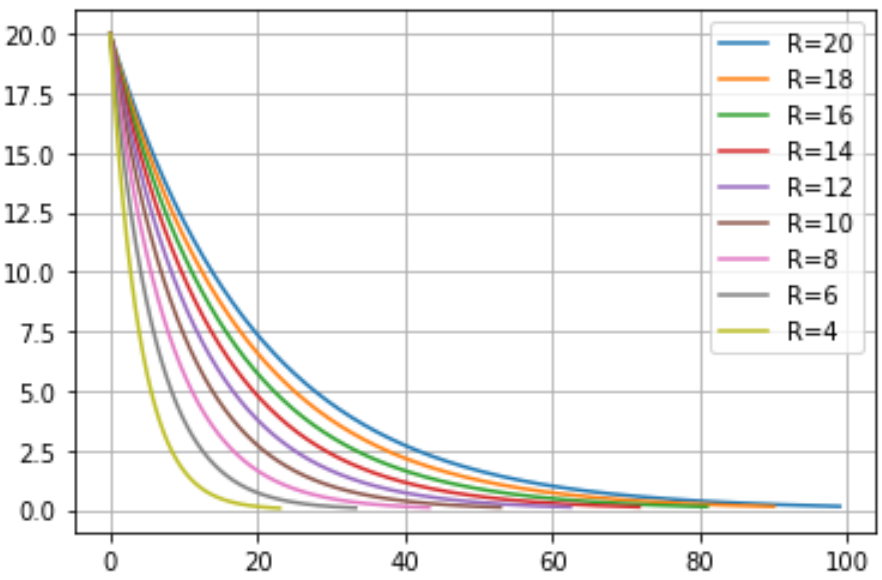


Рис. 27 Полученные графики

Как можно заметить из графиков, с уменьшением радиуса график становится более резким, и тело, пройдя меньшее расстояние, теряет большую часть своей скорости.

6.2. Исследование при различных значениях начальной скорости

Теперь посмотрим, что происходит с графиком при различных начальных скоростях. Значения скорости возьмем из диапазона v=range(60, 9, -5).

```
s = []
v = []
l = []
for v_n in range(60, 9 ,-5):
    s_t, v_t = runge_kutta_method(0.01, v_n, r)
    plt.plot(s_t, v_t)
    lab = 'v0='+str(v_n)
    l.append(lab)

plt.legend(l)
plt.grid()
```

Рис. 28 Код для исследования

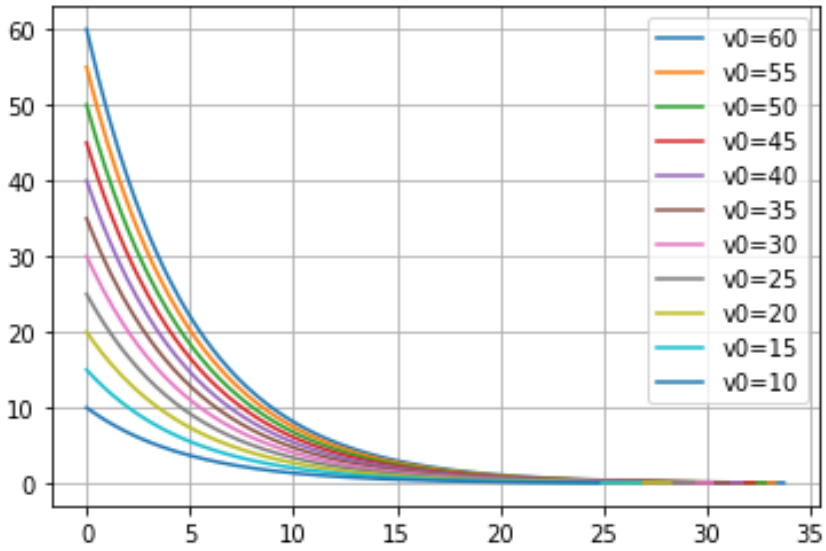


Рис. 29 Полученные графики

Как можно заметить из графиков, начальная скорость влияет на поведение тело вначале движения, пройдя некоторое расстояние, тело все равно практически полностью останавливается.

Вывод

В результате выполненной работы методом Эйлера и Рунге-Кутты 4-ого порядка были получены два численных решения ОДУ, имеющего физический смысл. Было показано, что оба решения хорошо согласуются с теорией.

Была определена зависимость ошибки от шага в виде таблицы. Были определены порядки точности обоих методов, которые согласуются с ожидаемыми (1 для метода Эйлера и 4 для метода Рунге-Кутты 4-ого порядка) методами логарифмических невязок и экстраполяцией Ричардсона.

Было исследовано поведение графика получаемого решения при различных значениях входных параметров (Радиус окружности, по которой двигается тело и его начальная скорость).