МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по курсовой работе

по дисциплине «Дифференциальные уравнения»

Тема: Математический маятник

Студент гр. 8383	______________________	Киреев К.А.
Студент гр. 8383	______________________	Муковский Д.В.
Преподаватель	______________________	Павлов Д.А.

Санкт-Петербург

2021

Задание

Реализовать численное интегрирование динамической системы (математический маятник) несколькими методами с различными параметрами.

Выполнение работы

Математический маятник - классический пример гармонического осциллятора.

Пусть он состоит из материальной точки массой m, подвешенной на конце невесомой нерастяжимой нити или лёгкого стержня длины L, как показано на рис. 1.

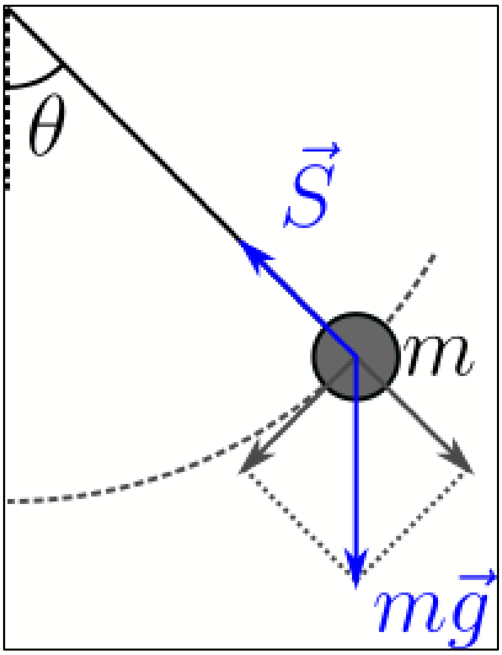

Рисунок 1 – Математический маятник

Маятник движется только в одной плоскости, и трение отсутствует.

Пусть θ обозначает угол между вертикальной осью и нитью, такой, что $\theta = 0$ означает положение равновесия. Предполагается, что потерь энергии в системе нет. Воспользуемся вторым законом Ньютона, чтобы найти дифференциальное уравнение, описывающее движение маятника. Пусть t обозначает время.

Движение маятника описывается уравнением:

$$L\ddot{\theta} = L\frac{d^2\theta}{dt^2} = -gsin(\theta)$$

Аналитическое решение

Уравнение не может быть решено аналитически. Однако в области малых углов $\sin(\theta) \approx \theta$ это уравнение может быть аппроксимировано как

$$L\ddot{\theta} \approx -g\theta,$$

которое имеет решение:

$$\theta = \theta_0 \cos(\omega t),$$

где θ_0 - начальная позиция при $t = 0$, а $\omega = \sqrt{g/L}$ - собственная частота колебаний. Период гармонических колебаний равен $T = 2\pi\ /\ \omega = 2\pi\sqrt{L/g}$

Аналитическое решение в программе:

```
def approx(t, theta0):
    return theta0*np.cos(np.sqrt(g/L)*t)
```

L = 50 – Длина стержня

g = 9.81 – Ускорение свободного падения

График аналитического решения уравнения с шагом $h = 0.001$ на $t = [0; 200]$:

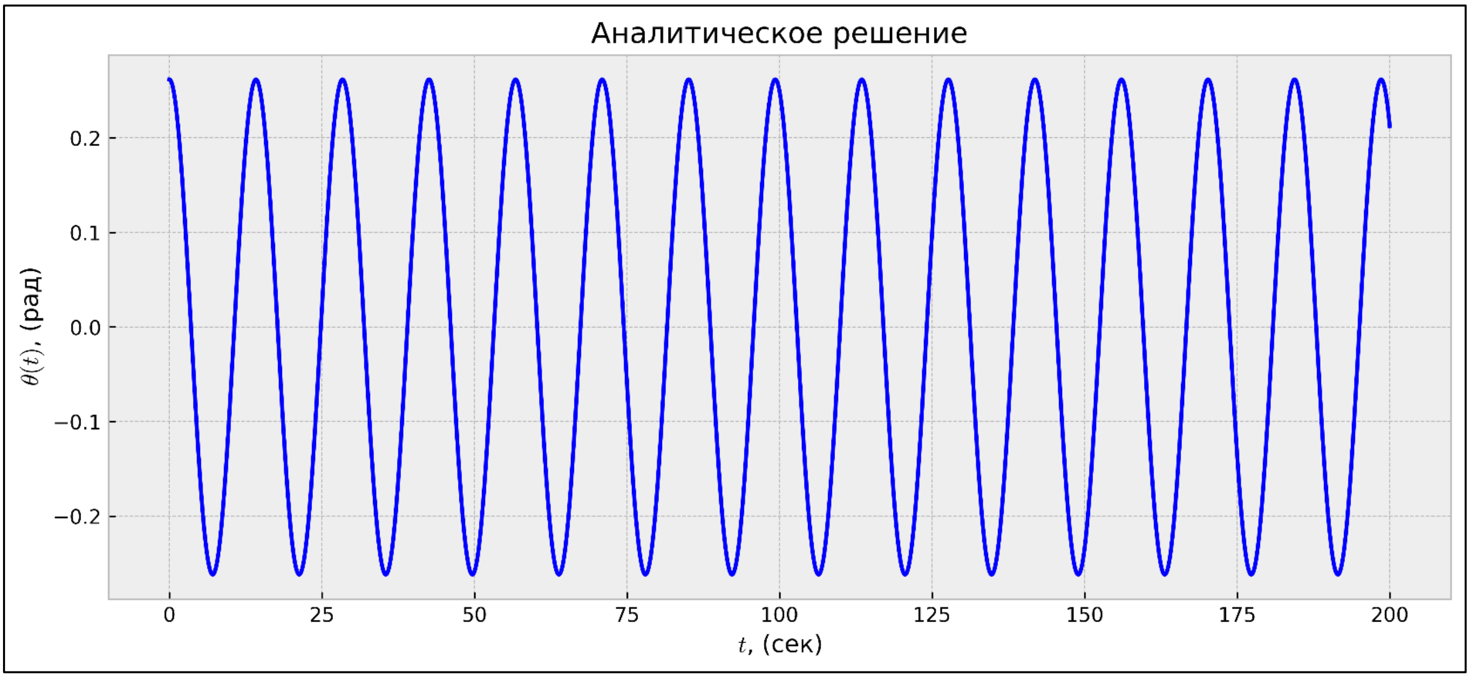

Рисунок 2 – Аналитическое решение

Численное решение

Запишем исходное дифференциальное уравнение в виде двух связанных дифференциальных уравнений первого порядка, введя угловую скорость $\omega = \dot{\theta}$:

$$d\theta = \omega\, dt$$

$$d\omega = -\frac{g}{L}\theta\, dt$$

Полученные уравнения в программе:

```
def f(X):
    theta = X[0]
    w = X[1]
    dtheta = w
    dw = -(g/L)*(theta)
    return np.array([dtheta, dw])
```

Явные одношаговые методы

- Метод Эйлера (1)

  Явный одношаговый метод семейства Рунге-Кутты первого порядка. Имеет погрешность на шаге $o(h^2)$ и погрешность в целом $o(h)$.

  Таблица Бутчера для метода:

$$\begin{array}{c|c} & 1 \\ \hline & 0 \end{array}$$

  Листинг метода:

```
def euler_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        X = X0 + h*k1
        t0 += h
        X0 = X
```

  График аналитического решения уравнения и решения методом Эйлера с шагом $h_1 = 0.001$ и $h_2 = 0.01$ представлен на рис. 3, метод вычислил значение функции f 199999 и 19999 раз соответственно на $t \in [0; 200]$.

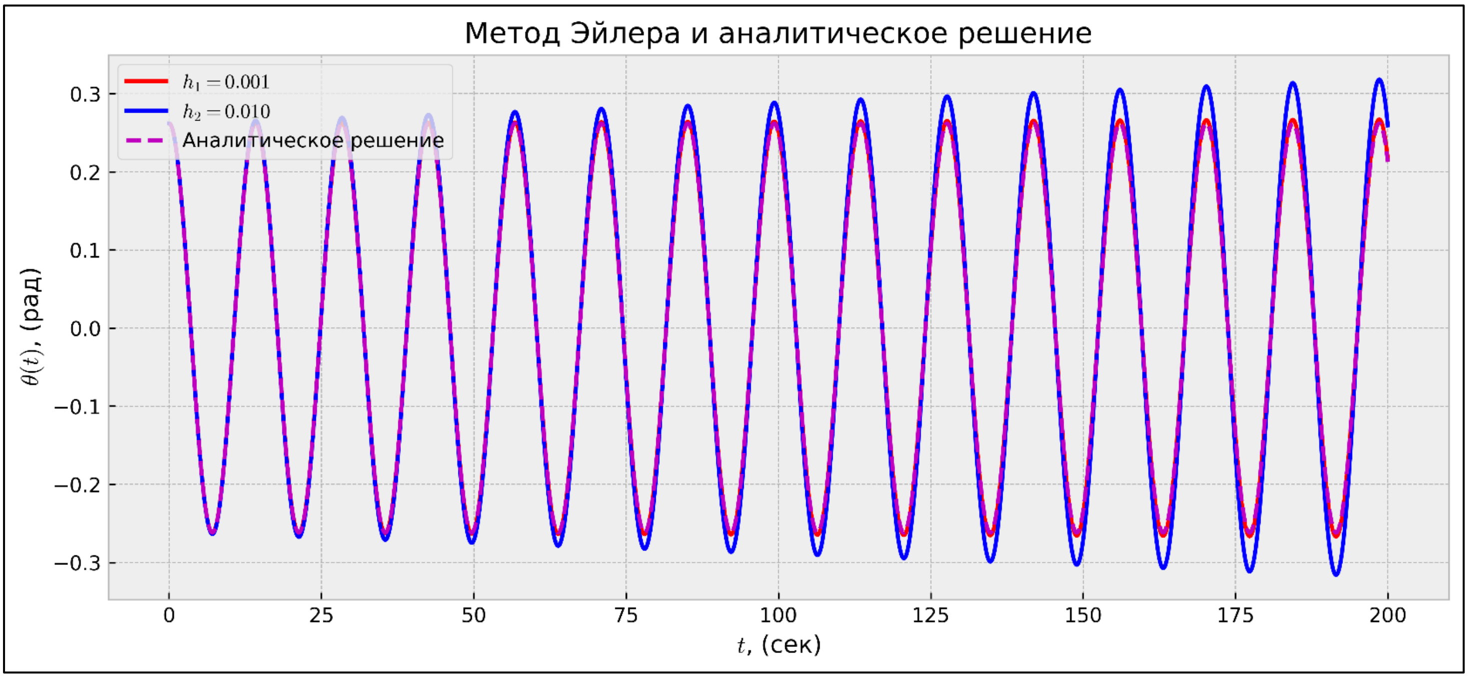

Рисунок 3 – Метод Эйлера и аналитическое решение

График модуля ошибки аналитического решения уравнения и решения методом Эйлера с шагом $h = 0.0001$ на $t \in [0; 200]$:

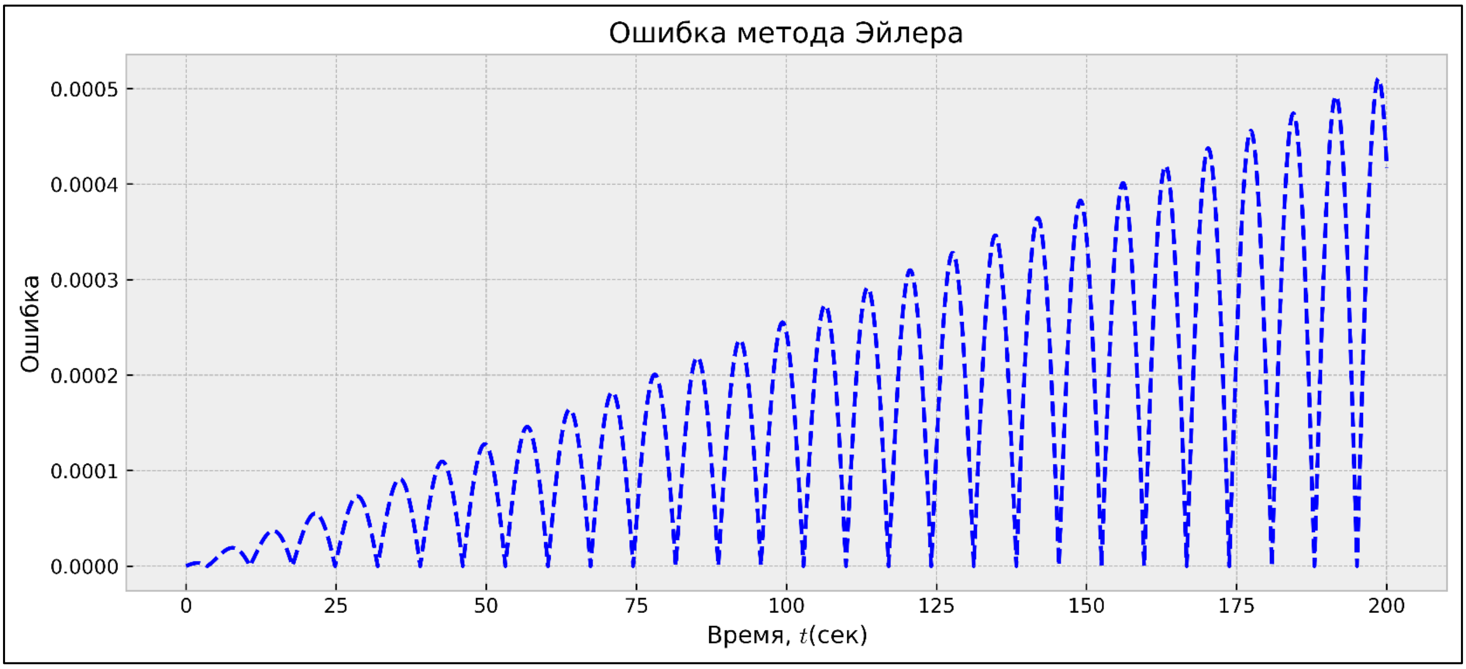

Рисунок 4 – Ошибка метода Эйлера

Максимальная ошибка равна ***0.00051056073400174150***

Видно, что ошибка быстро накапливается, а также, что метод неустойчив на больших временных промежутках и при большом значение шага, что можно увидеть на рис. 3.

- Метод средней точки (2)

  Явный одношаговый метод семейства Рунге-Кутты второго порядка. Имеет погрешность на шаге $o(h^3)$ и погрешность в целом $o(h^2)$.

  Таблица Бутчера для метода:

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 0 & \\ \hline & 0 & 1 \end{array}$$

  Листинг метода:

```
def midpoint_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        k2 = f(X0 + h/2*k1)
        X = X0 + h*k2
        t0 += h
        X0 = X
```

  График аналитического решения уравнения и решения методом средней точки с шагом $h_1 = 0.01$ и $h_2 = 0.001$ представлен на рис. 5, метод вычислил значение функции f 40000 и 399998 раз соответственно на $t \in [0; 200]$

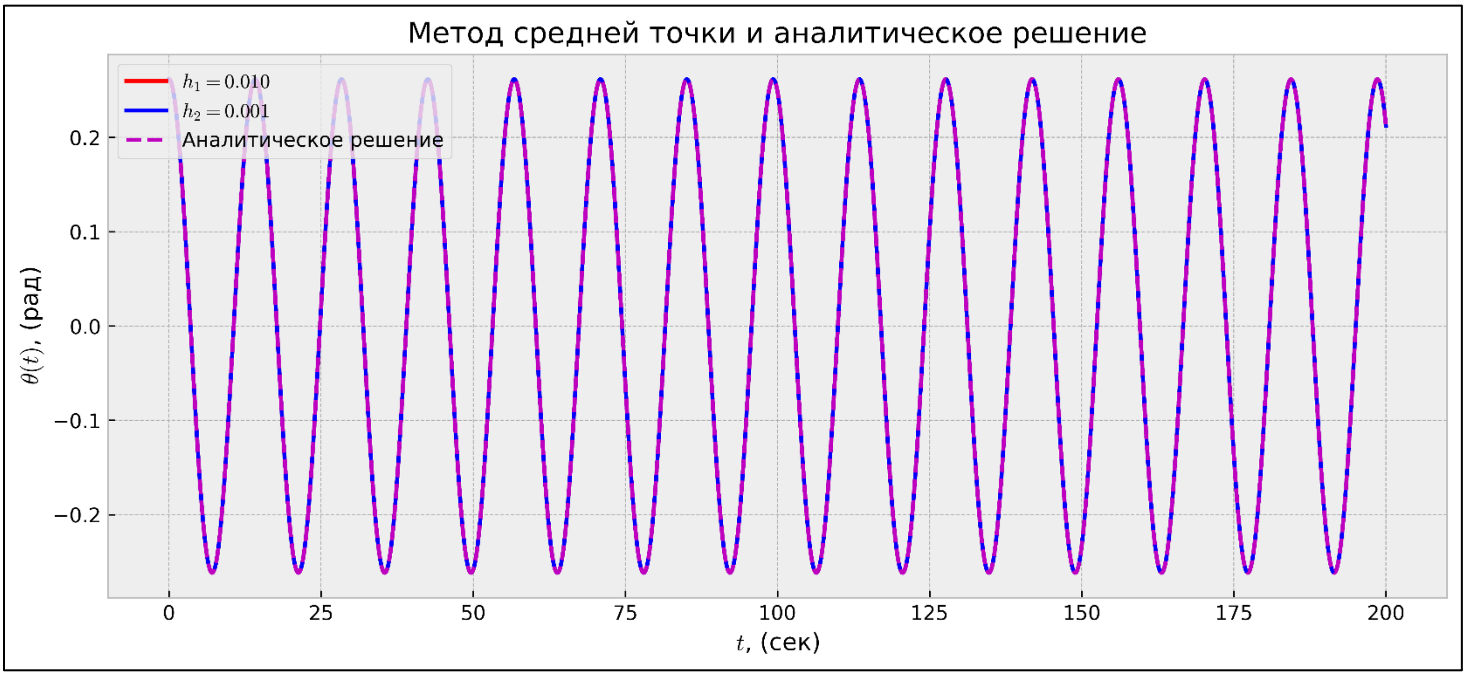

Рисунок 5 – Метод средней точки и аналитическое решение

График модуля ошибки аналитического решения уравнения и решения методом средней точки с шагом $h = 0.0001$ на $t \in [0; 200]$:

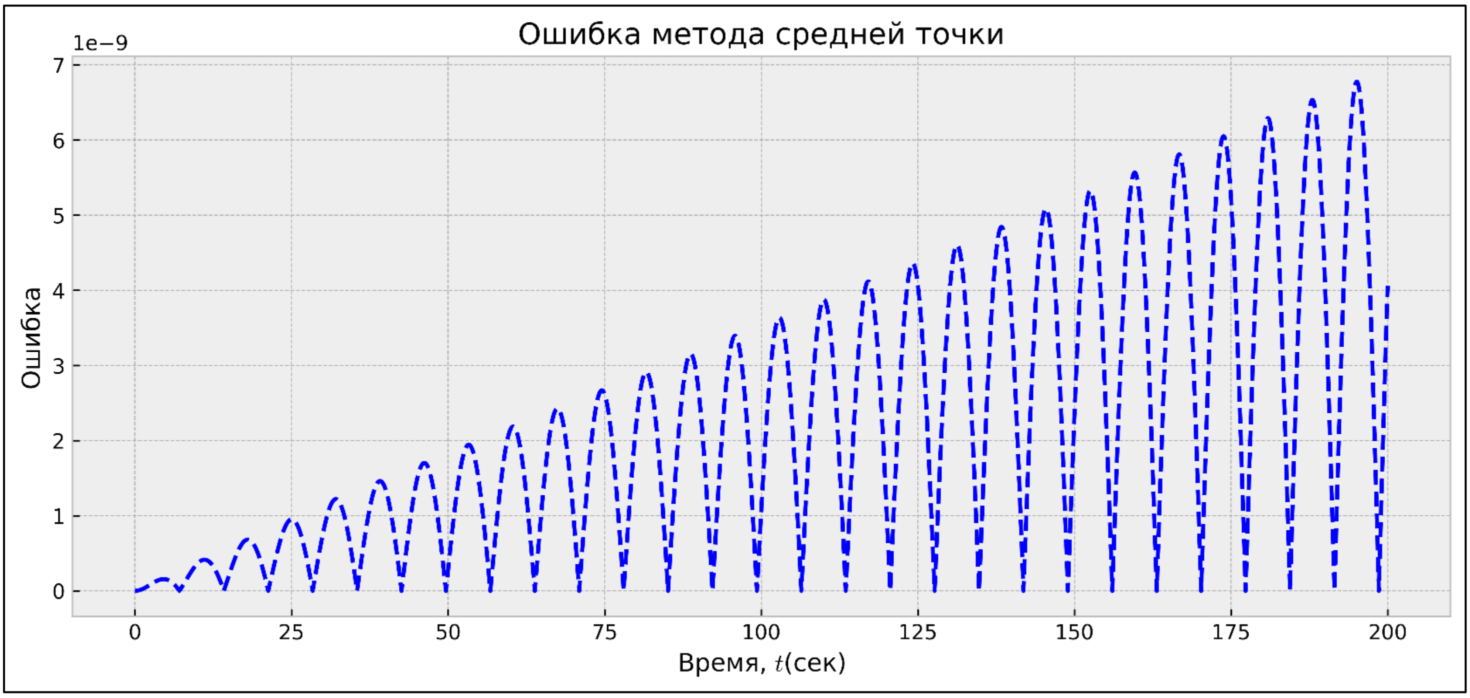

Рисунок 6 – Ошибка метода средней точки

Максимальная ошибка равна ***6.775920186052886e-09***

Метод выдает меньшую ошибку при вычислениях, чем метод Эйлера при таком же шаге. Так же можно заметить, что график метода и аналитического решения сливается, поэтому в последующих методах будет приводиться только ошибка методов.

- Метод Рунге-Кутты (4)

  Явный одношаговый метод семейства Рунге-Кутты второго порядка. Имеет погрешность на шаге $o(h^5)$ и погрешность в целом $o(h^4)$.

  Таблица Бутчера для метода:

$$
\begin{array}{c|cccc}
 & 0 & & & \\
 & 1/2 & 0 & & \\
 & 0 & 1/2 & 0 & \\
 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

  Листинг метода:

```
def RK4_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        k2 = f(X0 + h/2*k1)
        k3 = f(X0 + h/2*k2)
        k4 =  f(X0 + h*k3)
        X= X0 + h/6*(k1 + 2*k2 + 2*k3 + k4)
        t0 += h
        X0 = X
```

Метод вычислил значение функции f 80000 раз на $t \in [0; 200]$ с шагом $h = 0.01$

График модуля ошибки аналитического решения уравнения и решения методом Рунге-Кутты 4 с шагом $h = 0.01$:

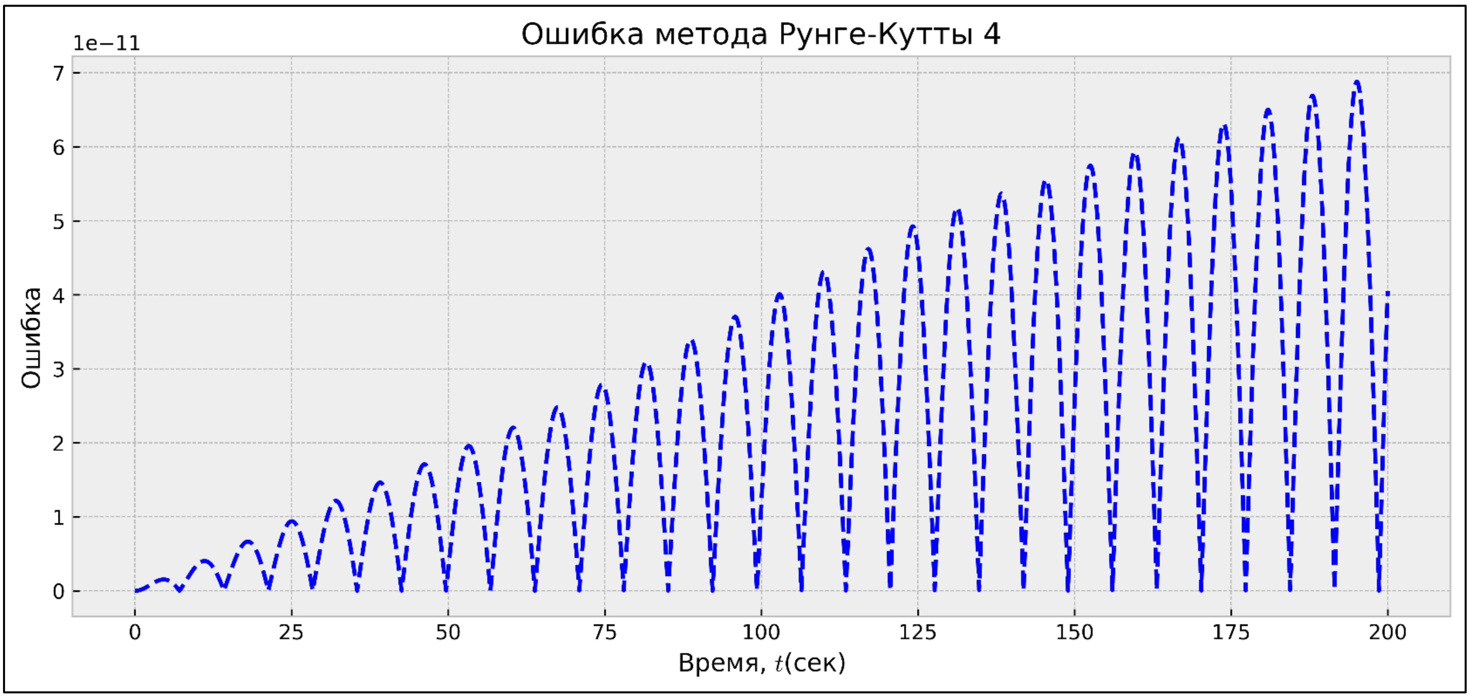

Рисунок 7 – Ошибка метода Рунге-Кутты 4

Максимальная ошибка равна ***6.879657260538963e-11***

В отличие от предыдущих методов более низких порядков данный метод даже на относительно крупном шаге выдает высокую точность.

Явные одношаговые вложенные методы

- Метод Богацкого-Шампина (3[2])

Метод Богацкого-Шампина — это метод Рунге-Кутты третьего порядка с четырьмя этапами со свойством FSAL, поэтому он использует примерно три вычисления функции на шаг. Также в методе реализован адаптивный шаг.

Таблица Бутчера для метода:

$$
\begin{array}{|cccc}
0 & & & \\
1/2 & 0 & & \\
0 & 3/4 & 0 & \\
2/9 & 1/3 & 4/9 & 0 \\
\hline
2/9 & 1/3 & 4/9 & 0 \\
\hline
7/24 & 1/4 & 1/3 & 1/8
\end{array}
$$

Листинг метода:

```
def BS_method(f, X0, t0, T0, h, call):
    errors, true_errors = [], []
    k1 = f(X0)
    while t0 < T0-h:
        call(X0, t0)
        k2 = f(X0 + h/2*k1)
        k3 = f(X0 + (3*h)/4*k2)
        X_new = X0 + (2*h)/9*k1 + (1*h)/3*k2 + (4*h)/9*k3
        k4 =  f(X_new)
        X_hat = X0 + h*(7*k1 + 6*k2 + 8*k3 + 3*k4)/24
        err = error_handler(X_new, X_hat)
        h = h * ((1/err)**(1/3))
        t0 += h
        eps = np.abs(X_new[0] - X_hat[0]) # оценка локальной ошибки
        true_eps = np.abs(true_sol(h, X0[0], X0[1]) - X_new[0]) # истинное значение локальной ошибки
        X0 = X_new
        k1 = k4
        errors.append(eps)
        true_errors.append(true_eps)
    return np.array(errors), np.array(true_errors)
```

Метод вычислил значение функции f 62152 раза на $t \in [0; 200]$ для $atol = 2e - 10, rtol = 2e - 10$.

На рис. 8 представлен график оценки локальной ошибки метода и истинного значения локальной ошибки. Оценка вычисляется как разность x и $\hat{x}$, а истинная локальная ошибка была получена как разность полученного в методе значения $x(t + h)$ и “точного” значения $x(t + h)$, рассчитанного аналитически с начальным условием $x(t)$.

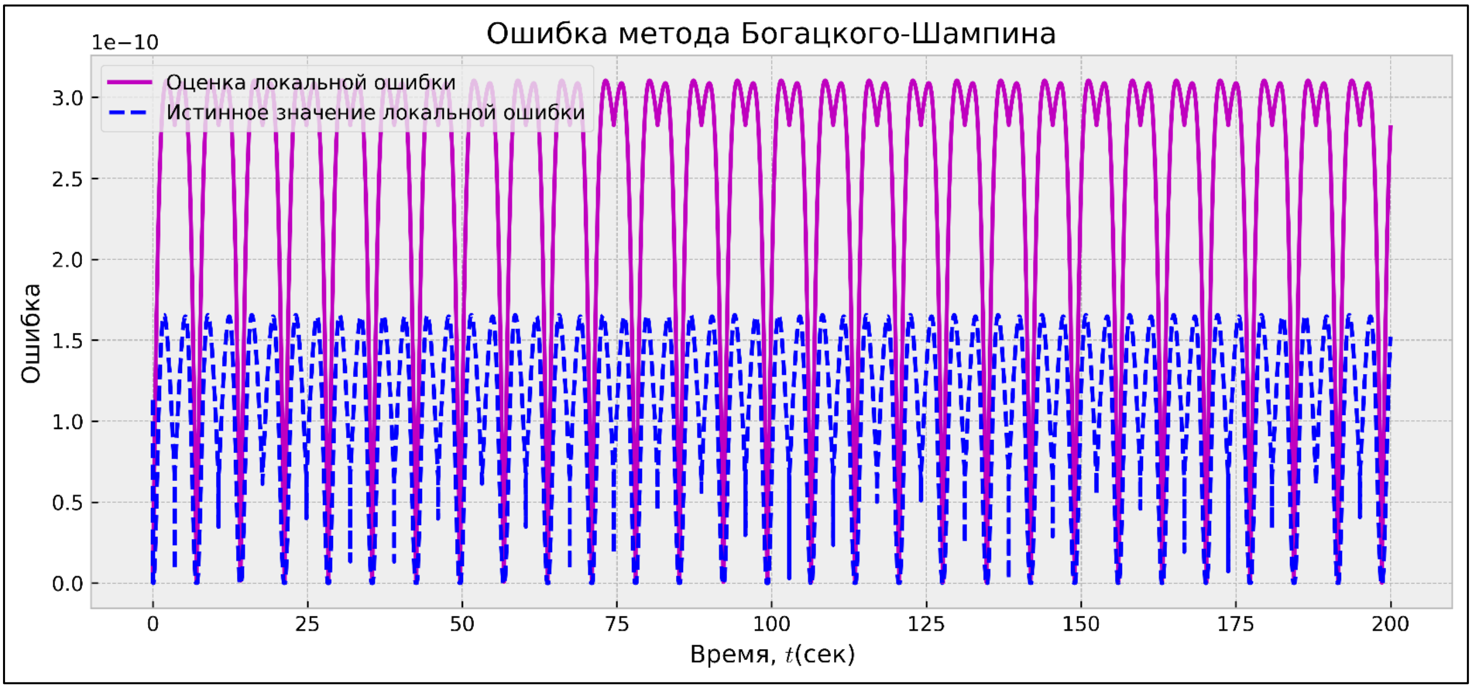

Рисунок 8 – Ошибка метода Богацкого-Шампина

Благодаря адаптивному подбору шага данный метод достигает высокой точности.

Многошаговые методы

- Метод Адамса-Башфорта

Формула для вычисления x_{n+1} выглядит следующим образом:

$$x_{n+1} = x_n + h(\frac{55}{24}f_n - \frac{59}{24}f_{n-1} + \frac{37}{24}f_{n-2} - \frac{9}{24}f_{n-3})$$

Для вычисления начальных значений был использован одношаговый метод Рунге-Кутты 4 порядка.

Листинг метода:

```
def AB_method(f, X0, t0, T0, h, call):
    RK4_method(f, X0, t0, t0+5*h, h, call)
    t0 = t0 + 3*h
    i = 3
    while t0 < T0-h:
        Y = X[i] + h/24. * (55*f(X[i]) - 59*f(X[i-1]) + 37*f(X[i-2]) - 9*f(X[i-3]))
        t0 = t0 + h
        i += 1
        call(Y, t0)
```

Метод вычислил значение функции f 80004 раза.

График модуля ошибки аналитического решения уравнения и решения методом Адамса-Башфорта представлен на рис. 9.

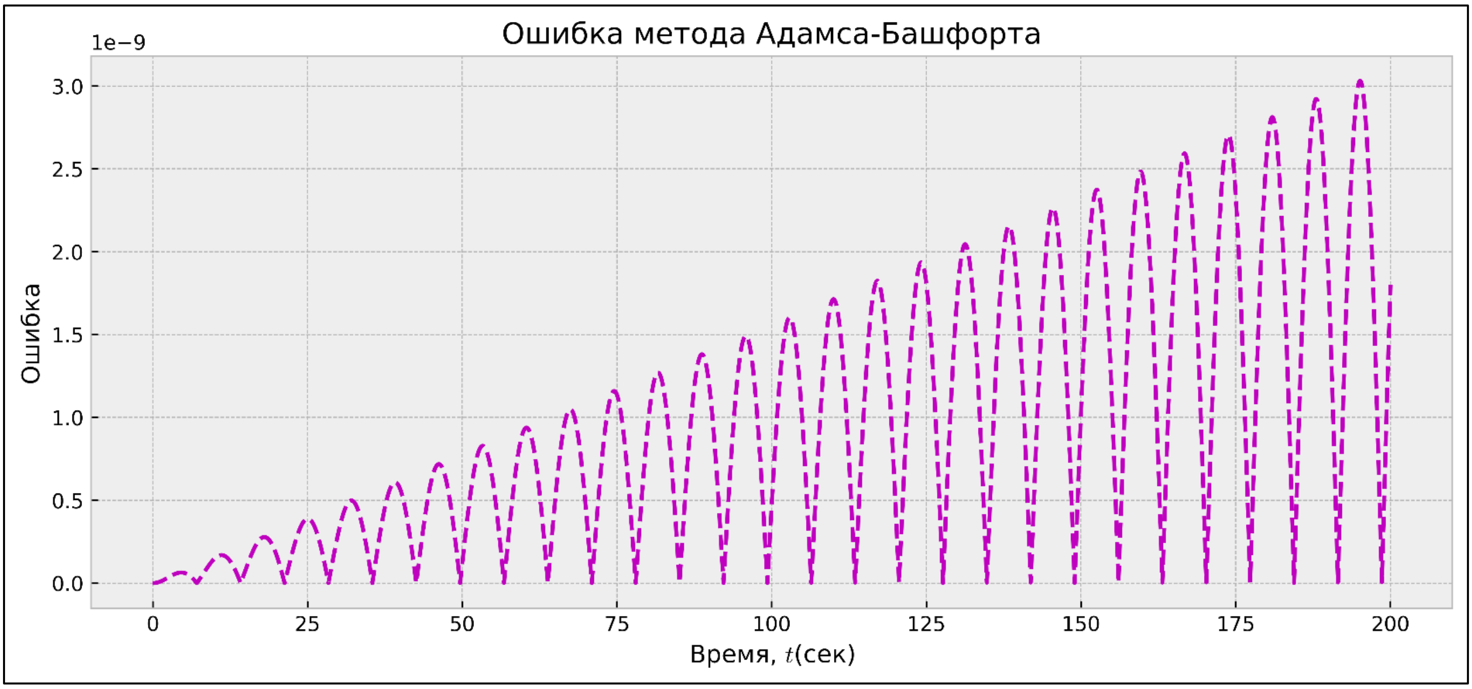

Рисунок 9 – Ошибка метода Адамса-Башфорта

Максимальная ошибка равна ***3.03143439170372445e-09***

- Метод Адамса-Мултона

$$x_{n+1} \leftarrow x_n + h\sum_{i=0}^{k-1}\gamma_i \nabla^i f_n - Predictor(P)$$

$$f_{n+1} \leftarrow f(x_{n+1}) - Evaluator(PE)$$

$$x_{n+1} \leftarrow x_n + h\sum_{i=0}^{k}\gamma_i^* \nabla^i f_{n+1} - Corrector(PEC)$$

В качестве разгона выступает метод Рунге-Кутты 4 порядка. В качестве предиктора выступает метод Адамса-Башфорта. Используется схема $PECE$.

```
def AM_method(f, X0, t0, T0, h, call):
    RK4_method(f, X0, t0, t0+5*h, h, call)
    k0, k1, k2, k3 = f(X[3]), f(X[2]), f(X[1]), f(X[0])
    t0 = t0 + 3*h
    i = 3
    while t0 < T0-h:
        k4, k3, k2, k1 = k3, k2, k1, k0
        # Предиктор (Адамс-Башфорт)
        Y = X[i] + h/24. * (55*k1 - 59*k2 + 37*k3 - 9*k4)
        k0 = f(Y)
        # Корректор (Адамс-Мултон)
        Y = X[i] + h/24. * (9*k0 + 19*k1 - 5*k2 + 1*k3)
        k0 = f(Y)
        t0 += h
        i += 1
        call(Y, t0)
```

Метод вычислил значение функции f 40014 раз.

График модуля ошибки аналитического решения уравнения и решения методом Адамса-Мултона представлен на рис. 10.

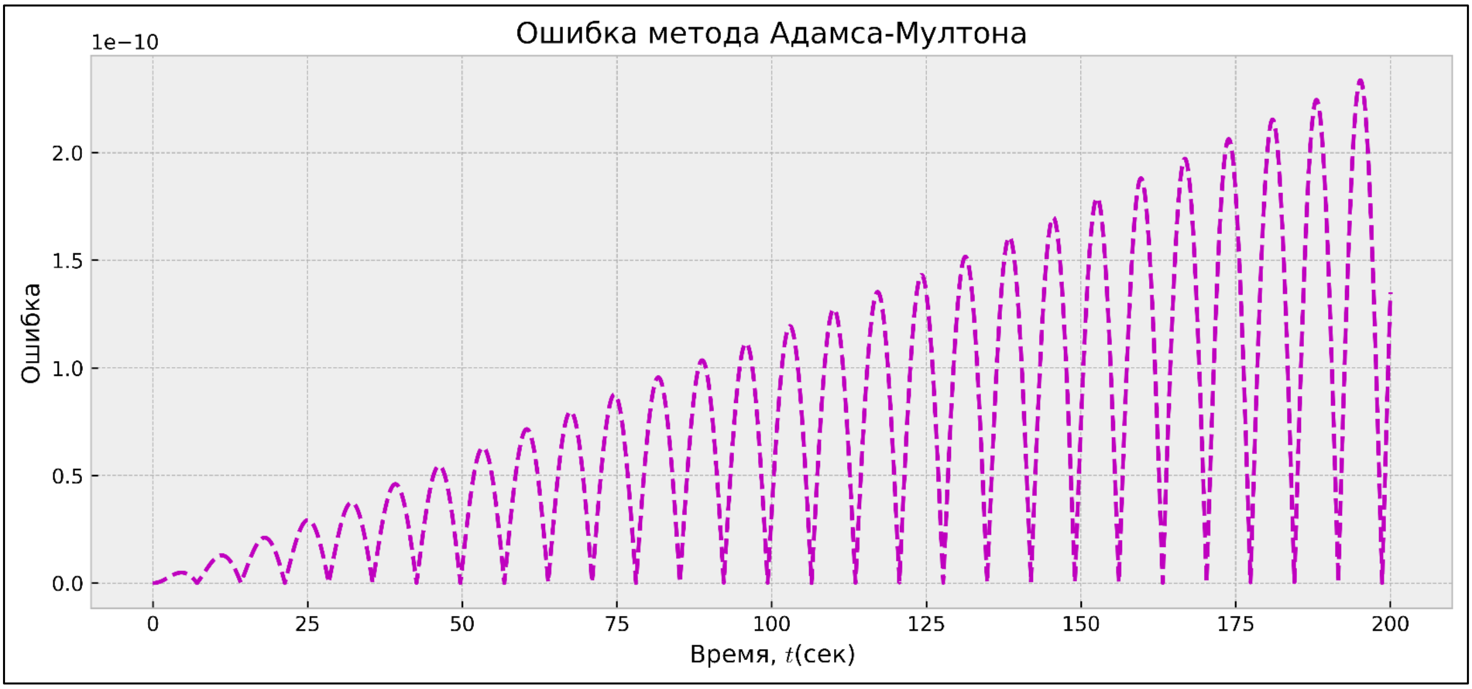

Рисунок 10 – Ошибка метода Адамса-Мултона

Максимальная ошибка равна ***2.336041429740199e-10***

Сравнение аналитического решения и scipy.odeint

Было выполнено сравнение аналитического решения задачи и решения, полученного с помощью scipy.odeint на $t \in [0; 200]$ для $atol = 2e - 10, rtol = 2e - 10$. Метод scipy.odeint вычислил значение функции f 1691 раз.

График модуля ошибки аналитического решения уравнения и решения scipy.odeint для $atol = 2e - 10, rtol = 2e - 10$:

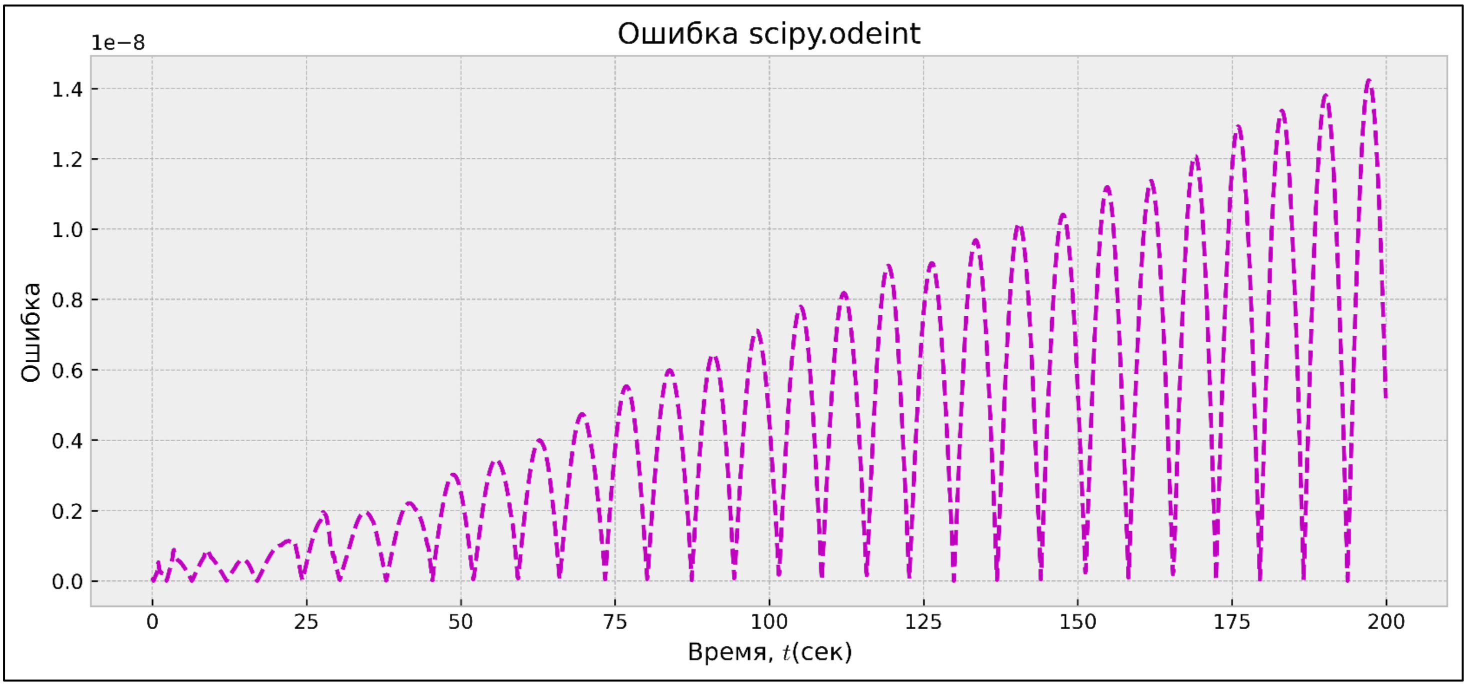

Рисунок 12 – Ошибка scipy.odeint

Максимальная ошибка равна ***1.4221851585283218e-08***

Сравнение методов

Цель: получить решение задачи с заданной точностью как можно реже вызывая функцию f.

Решим задачу с точностью $2e-10$ и определим какое количество вызовов функции f необходимо методам. Сравнение проводилось на временном промежутке $t \in [0; 20]$

Метод	***Шаг***	***Количество вызовов функции f***
Эйлера	*Очень маленький*	*>1000000*
Средней точки	*5.0753e-05*	*788132*
Рунге-Кутты 4	*0.022876*	*3496*
Богацкого-Шампина	*Адаптивный шаг*	*6259*
Адамса-Башфорта	*0.008862938119652 51*	*8124*
Адамса-Мултона	*0.01667718169966658*	*2412*

Метод Эйлера проигрывает любому из методов, так как количество вызовов функции f намного больше в сравнении с другими методами. Также относительно плохие результаты показывает метод средней точки, хотя количество вызовов функции намного меньше метода Эйлера.

Метод Адамса-Мултона показал самое низкое количество вызовов $f = 2412$ для заданной точности $2e - 10$, что делает его предпочтительным для решения данной задачи. Также хорошие результаты показал метод Рунге-Кутты 4 порядка, который вызвал функцию f 3496 раз. Остальные методы показали средние результаты.

Переменный шаг

Рассмотрим работу методов Адамса-Башфорта и Адамса-Мултона с различной длиной шага.

- Сравним метод Адамса-Башфорта с длинами шага $h_1 = 0.01, h_2 = 0.001$ и $h_3 = 0.0001$ на временном промежутке $t \in [0; 20]$

Шаг	***Количество вызовов f***	***Максимальная ошибка***
0.01	*8000*	*2.7792e-10*
0.001	*80000*	*2.6995e-13*
0.0001	*800004*	*3.3038e-12*

Можно заметить, что при шаге *0.0001* максимальная ошибка стала хуже, чем при шаге *0.001.* Это называется *численное насыщение*.

Теперь определим какое количество вызовов функции f необходимо методу Адамса-Башфорта, чтобы не наблюдалось эффекта численного насыщения.

Шаг	***Количество вызовов f***	***Максимальная ошибка***
0.0007855167211278952	*101844*	*4.974076706076858e-13*
0.0007069650490151057	*113160*	*1.2229522949880334e-12*

Видно, что численное насыщение начинает проявляться при количестве вызовов функции $f_{call} \in [101844; 113160]$

- Проделаем те же действия для метода Адамса-Мултона. Сравним метод с длинами шага $h_1 = 0.1, h_2 = 0.01, h_3 = 0.001$ и $h_4 = 0.0001$ на временном промежутке $t \in [0; 20]$

Шаг	***Количество вызовов f***	***Максимальная ошибка***
0.1	*412*	*2.1656e-07*
0.01	*4012*	*2.1081e-11*
0.001	*40012*	*2.8982e-13*
0.0001	*400014*	*3.3037e-12*

Опять же можно заметить, что наблюдается численное насыщение.

Определим какое количество вызовов функции f необходимо методу Адамса-Мултона, чтобы не наблюдалось эффекта численного насыщения.

Шаг	***Количество вызовов f***	***Максимальная ошибка***
0.0007855167211278952	*50934*	*4.880262860496032e-13*
0.0007069650490151057	*56592*	*1.2164089180366489e-12*

Видно, что численное насыщение начинает проявляться при количестве вызовов функции $f_{call} \in [50394; 56592]$

Выводы

В ходе курсовой работы были реализованы 6 различных алгоритмов решения системы ОДУ. При заданной цели получения решения с заданной точностью при наименьшем количестве вызовов функции f лучше всего себя показал метод Адамса-Мултона. В целом же при увеличении порядка метода либо же при уменьшении длины шага можно добиться увеличения точности решения задачи.

Приложение А. Исходный код.

```
import numpy as np
import pandas as pd
from scipy.integrate import odeint
import matplotlib.pyplot as plt
plt.style.use('bmh')
figsize = (12, 5)
dpi = 600


g = 9.81
L = 50
theta0 = np.pi/12
t0 = 0
T0 = 200
X, T = [], []
f_counter = []

def approx(t, theta0):
    return theta0*np.cos(np.sqrt(g/L)*t)


t = np.arange(t0, T0, 0.001)
plt.figure(figsize=figsize, dpi=dpi)
plt.title("Аналитическое решение")
plt.plot(t, approx(t, theta0), "b-")
plt.xlabel(r"$t$, (сек)")
plt.ylabel(r"$\theta(t)$, (рад)")
plt.show()


def f(x):
    theta = x[0]
    w = x[1]
    dtheta = w
    dw = -(g/L)*(theta)
    f_counter.append(1)
    return np.array([dtheta, dw])


def callback(x, t):
    T.append(t)
    X.append(x.copy())

def out(x, t):
    return np.array(x)[:, 0], np.array(x)[:, 1], np.array(t)

def euler_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        X = X0 + h*k1
```

```
        t0 += h
        X0 = X

h1 = 0.001
X, T, f_counter = [], [], []
euler_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))

h2 = 0.01
X, T, f_counter = [], [], []
euler_method(f, [theta0, 0], t0, T0, h2, callback)
theta2, w2, t2 = out(X, T)
print(len(f_counter))

plt.figure(figsize=figsize, dpi=dpi)
plt.title("Метод Эйлера и аналитическое решение")
plt.plot(t1, theta1, "r-", label=r"$h_1=%.3f$"%(h1))
plt.plot(t2, theta2, "b-", label=r"$h_2=%.3f$"%(h2))
plt.plot(t1, approx(t1, theta0), "m--", label=r"Аналитическое решение")
plt.xlabel(r"$t$, (сек)")
plt.ylabel(r"$\theta(t)$, (рад)")
plt.legend(loc='upper left')
plt.show()

he = 0.0001
X, T = [], []
euler_method(f, [theta0, 0], t0, T0, he, callback)
thetae, we, te = out(X, T)
error = np.abs(approx(te, theta0) - thetae)
plt.figure(figsize=figsize, dpi=dpi)
plt.plot(te, error, "b--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода Эйлера")
plt.show()
print("Максимальная ошибка равна ", np.max(error))


def midpoint_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        k2 = f(X0 + h/2*k1)
        X = X0 + h*k2
        t0 += h
        X0 = X

h1 = 0.01
X, T, f_counter = [], [], []
midpoint_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))
h2 = 0.001
X, T, f_counter = [], [], []
midpoint_method(f, [theta0, 0], t0, T0, h2, callback)
```

```
theta2, w2, t2 = out(X, T)
print(len(f_counter))

plt.figure(figsize=figsize, dpi=dpi)
plt.title("Метод средней точки и аналитическое решение")
plt.plot(t1, theta1, "r-", label=r"$h_1=%.3f$"%(h1))
plt.plot(t2, theta2, "b-", label=r"$h_2=%.3f$"%(h2))
plt.plot(t1, approx(t1, theta0), "m--", label=r"Аналитическое решение")
plt.xlabel(r"$t$, (сек)")
plt.ylabel(r"$\theta(t)$, (рад)")
plt.legend(loc='upper left')
plt.show()


he = 0.0001
X, T = [], []
midpoint_method(f, [theta0, 0], t0, T0, he, callback)
thetae, we, te = out(X, T)
error = np.abs(approx(te, theta0) - thetae)
plt.figure(figsize=figsize, dpi=dpi)
plt.plot(te, error, "b--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода средней точки")
plt.show()
print("Максимальная ошибка равна ", np.max(error))


def RK4_method(f, X0, t0, T0, h, call):
    while t0 < T0-h:
        call(X0, t0)
        k1 = f(X0)
        k2 = f(X0 + h/2*k1)
        k3 = f(X0 + h/2*k2)
        k4 =  f(X0 + h*k3)
        X= X0 + h/6*(k1 + 2*k2 + 2*k3 + k4)
        t0 += h
        X0 = X

h1 = 0.01
X, T, f_counter = [], [], []
RK4_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))

plt.figure(figsize=figsize, dpi=dpi)
plt.title("Метод Рунге-Кутты 4 и аналитическое решение")
plt.plot(t1, theta1, "m", label=r"$h_1=%.3f$"%(h1))
plt.plot(t1, approx(t1, theta0), "b--", label=r"Аналитическое решение")
plt.xlabel(r"$t$, (сек)")
plt.ylabel(r"$\theta(t)$, (рад)")
plt.legend(loc='upper left')
plt.show()

he = 0.01
X, T = [], []
```

```
RK4_method(f, [theta0, 0], t0, T0, he, callback)
thetae, we, te = out(X, T)
error = np.abs(approx(te, theta0) - thetae)
plt.figure(figsize=figsize, dpi=dpi)
plt.plot(te, error, "b--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода Рунге-Кутты 4")
plt.show()
print("Максимальная ошибка равна ", np.max(error))


def true_sol(t, x0, w0):
    return x0 * np.cos(np.sqrt(g/L) * t) + (w0/np.sqrt(g/L)) * np.sin(np.sqrt(g/L)
* t)

def error_handler(x, x_hat):
    err = 0
    x = np.array(x)
    x_hat = np.array(x_hat)
    tol = np.zeros(len(x))
    for i in range(len(x)):
        tol[i] = atol + max(abs(x[i]), abs(x_hat[i])) * rtol
        err += ((x_hat[i] - x[i]) / tol[i]) ** 2
    err = np.sqrt(err/(len(x)))
    return err

def BS_method(f, X0, t0, T0, h, call):
    errors, true_errors = [], []
    k1 = f(X0)
    while t0 < T0-h:
        call(X0, t0)
        k2 = f(X0 + h/2*k1)
        k3 = f(X0 + (3*h)/4*k2)
        X_new = X0 + (2*h)/9*k1 + (1*h)/3*k2 + (4*h)/9*k3
        k4 =  f(X_new)
        X_hat = X0 + h*(7*k1 + 6*k2 + 8*k3 + 3*k4)/24
        err = error_handler(X_new, X_hat)
        h = h * ((1/err)**(1/3))
        t0 += h
        eps = np.abs(X_new[0] - X_hat[0]) # оценка локальной ошибки
        true_eps  =  np.abs(true_sol(h,  X0[0],  X0[1])  -  X_new[0])  #  истинное
значение локальной ошибки
        X0 = X_new
        k1 = k4
        errors.append(eps)
        true_errors.append(true_eps)
    return np.array(errors), np.array(true_errors)

atol, rtol = 2e-10, 2e-10
h1 = 0.01
T0 = 20
X, T, f_counter = [], [], []
e1, te1 = BS_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))
```

```
plt.figure(figsize=figsize, dpi=dpi)
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода Богацкого-Шампина")
plt.plot(t1, e1, "m-", label=r'Оценка локальной ошибки')
plt.plot(t1, te1*2e-4, "b--", label=r'Истинное значение локальной ошибки')
plt.legend(loc='upper left')
plt.show()


def AB_method(f, X0, t0, T0, h, call):
    RK4_method(f, X0, t0, t0+5*h, h, call)
    t0 = t0 + 3*h
    i = 3
    while t0 < T0-h:
        Y = X[i] + h/24. * (55*f(X[i]) - 59*f(X[i-1]) + 37*f(X[i-2]) - 9*f(X[i-
3]))
        t0 = t0 + h
        i += 1
        call(Y, t0)

h1 = 0.01
X, T, f_counter = [], [], []
AB_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))

plt.figure(figsize=figsize, dpi=dpi)
plt.plot(t1, np.abs(theta1-approx(t1, theta0)), "m--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода Адамса-Башфорта")
plt.show()
print("Максимальная ошибка равна ", np.max(np.abs(theta1-approx(t1, theta0))))


def AM_method(f, X0, t0, T0, h, call):
    RK4_method(f, X0, t0, t0+5*h, h, call)
    k0, k1, k2, k3 = f(X[3]), f(X[2]), f(X[1]), f(X[0])
    t0 = t0 + 3*h
    i = 3
    while t0 < T0-h:
        k4, k3, k2, k1 = k3, k2, k1, k0
        # Предиктор (Адамс-Башфорт)
        Y = X[i] + h/24. * (55*k1 - 59*k2 + 37*k3 - 9*k4)
        k0 = f(Y)
        # Корректор (Адамс-Мултон)
        Y = X[i] + h/24. * (9*k0 + 19*k1 - 5*k2 + 1*k3)
        k0 = f(Y)
        t0 += h
        i += 1
        call(Y, t0)

h1 = 0.01
X, T, f_counter = [], [], []
```

```
AM_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(len(f_counter))

plt.figure(figsize=figsize, dpi=dpi)
plt.plot(t1, np.abs(theta1-approx(t1, theta0)), "m--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка метода Адамса-Мултона")
plt.show()
print("Максимальная ошибка равна ", np.max(np.abs(theta1-approx(t1, theta0))))


def fsp(X, t):
    f_counter.append(1)
    return np.array([X[1], -(g/L)*(X[0])])
t = np.arange(t0, T0, 0.1)
f_counter = []
tol = 2e-10
Y = odeint(fsp, [theta0, 0], t, atol=tol, rtol=tol)
plt.figure(figsize=figsize, dpi=dpi)
plt.plot(t, np.abs(Y[:, 0] - approx(t, theta0)), "m--")
plt.xlabel("Время, $t$(сек)")
plt.ylabel("Ошибка")
plt.title("Ошибка scipy.odeint")
plt.show()
print(len(f_counter))
print(np.max(np.abs(Y[:, 0] - approx(t, theta0))))


methods = [midpoint_method, RK4_method, AB_method, AM_method]

T0 = 20
h = 0.1
X, T, f_counter = [], [], []
for method in methods:
    cur_h = h
    print(f'Для метода "{method.__name__}"')
    method(f, [theta0, 0], t0, T0, cur_h, callback)
    theta1, w1, t1 = out(X, T)
    print(f'h = {cur_h} | err = {np.max(np.abs(approx(t1, theta0) - theta1))}')
    while np.max(np.abs(approx(t1, theta0) - theta1)) > 2e-10:
        cur_h *= 0.9
        X, T, f_counter = [], [], []
        method(f, [theta0, 0], t0, T0, cur_h, callback)
        theta1, w1, t1 = out(X, T)
        print(f'h = {cur_h} | err = {np.max(np.abs(approx(t1, theta0) - theta1))}
| f_call = {len(f_counter)}')
T0 = 200


T0 = 20
h1 = 0.01
X, T, f_counter = [], [], []
AB_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
```

```
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t1, theta0) - theta1)
print(f"Максимальная ошибка равна {np.max(error)}\n")
h1 = 0.001
X, T, f_counter = [], [], []
AB_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t1, theta0) - theta1)
print(f"Максимальная ошибка равна {np.max(error)}\n")
h3 = 0.0001 # численное насыщение
X, T, f_counter = [], [], []
AB_method(f, [theta0, 0], t0, T0, h3, callback)
theta3, w3, t3 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t3, theta0) - theta3)
print(f"Максимальная ошибка равна {np.max(error)}\n")


h1 = 0.1
X, T, f_counter = [], [], []
AM_method(f, [theta0, 0], t0, T0, h1, callback)
theta1, w1, t1 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t1, theta0) - theta1)
print(f"Максимальная ошибка равна {np.max(error)}\n")
h2 = 0.01
X, T, f_counter = [], [], []
AM_method(f, [theta0, 0], t0, T0, h2, callback)
theta2, w2, t2 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t2, theta0) - theta2)
print(f"Максимальная ошибка равна {np.max(error)}\n")
h3 = 0.001
X, T, f_counter = [], [], []
AM_method(f, [theta0, 0], t0, T0, h3, callback)
theta3, w3, t3 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t3, theta0) - theta3)
print(f"Максимальная ошибка равна {np.max(error)}\n")
h4 = 0.0001
X, T, f_counter = [], [], []
AM_method(f, [theta0, 0], t0, T0, h4, callback)
theta4, w4, t4 = out(X, T)
print(f"Вызовы функции f: {len(f_counter)}")
error = np.abs(approx(t4, theta0) - theta4)
print(f"Максимальная ошибка равна {np.max(error)}\n")

methods = [AM_method]
T0 = 20
h = 0.1
f_err = 1
X, T, f_counter = [], [], []
for method in methods:
    cur_h = h
    cur_err = f_err
```

```
        print(f'Для метода "{method.__name__}"')
        method(f, [theta0, 0], t0, T0, cur_h, callback)
        theta1, w1, t1 = out(X, T)
        print(f'h = {cur_h} | err = {np.max(np.abs(approx(t1, theta0) - theta1))}')
        while np.max(np.abs(approx(t1, theta0) - theta1)) > 2e-10:
            cur_err = np.max(np.abs(approx(t1, theta0) - theta1))
            cur_h *= 0.9
            X, T, f_counter = [], [], []
            method(f, [theta0, 0], t0, T0, cur_h, callback)
            theta1, w1, t1 = out(X, T)
            print(f'h = {cur_h} | err = {np.max(np.abs(approx(t1, theta0) - theta1))}
| f_call = {len(f_counter)}')
```