**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, подвешенной на конце невесомой нерастяжимой нити или лёгкого стержня длины , как показано на рис. 1.

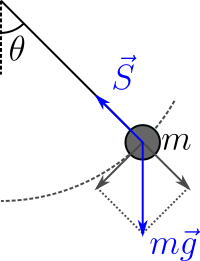


Рисунок 1 – Математический маятник

Маятник движется только в одной плоскости, и трение отсутствует.

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

Движение маятника описывается уравнением:

***Аналитическое решение***

Уравнение не может быть решено аналитически. Однако в области малых углов  это уравнение может быть аппроксимировано как

которое имеет решение:

где  - начальная позиция при , а  - собственная частота колебаний.

Период гармонических колебаний равен

Аналитическое решение в программе:

**

*L* = 50 – Длина стержня

*g* = 9.81 – Ускорение свободного падения

График аналитического решения уравнения с шагом на :

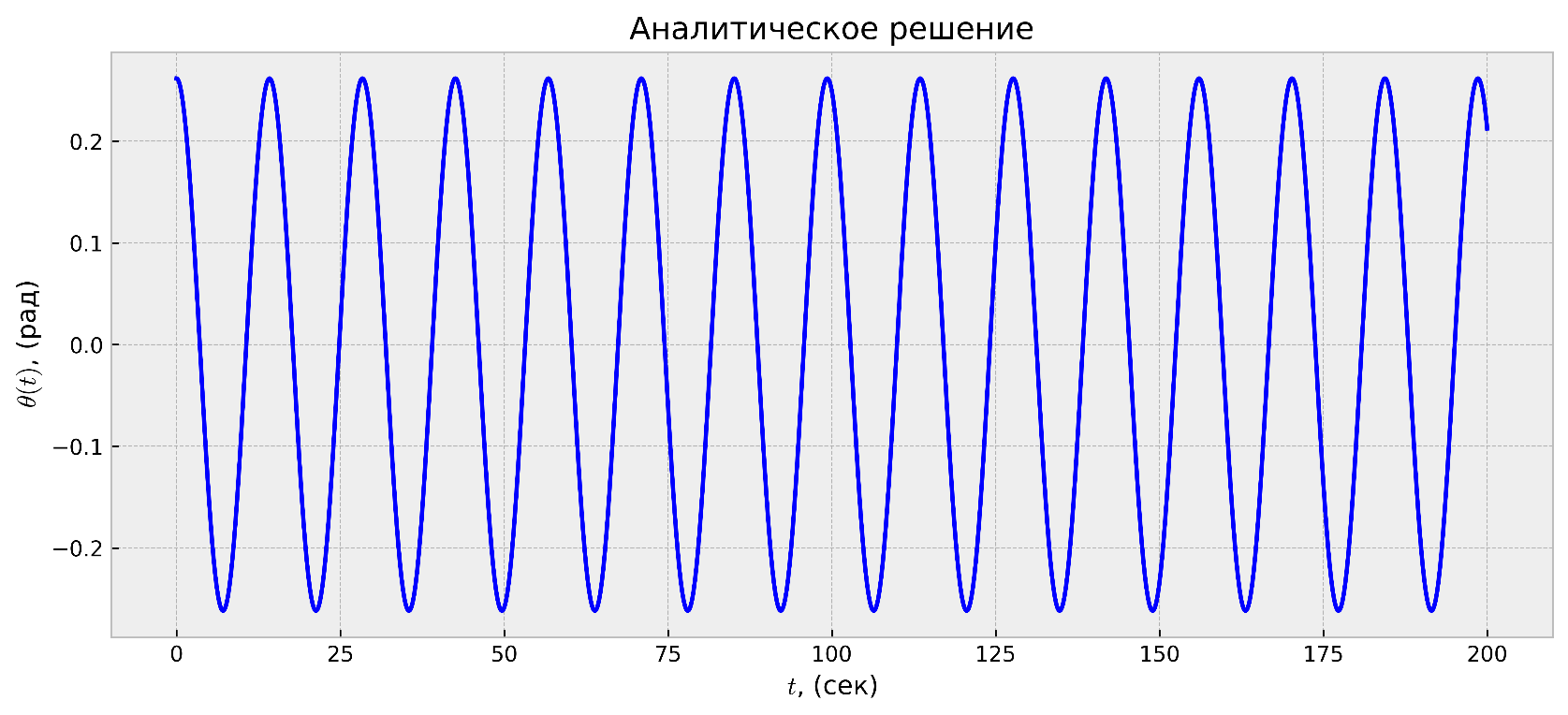
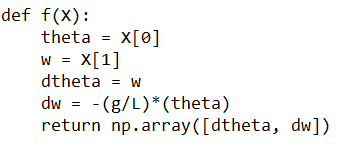


Рисунок 2 – Аналитическое решение

***Численное решение***

Запишем исходное дифференциальное уравнение в виде двух связанных дифференциальных уравнений первого порядка, введя угловую скорость :

Полученные уравнения в программе:

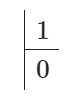


***Явные одношаговые методы***

* Метод Эйлера (1)

Явный одношаговый метод семейства Рунге-Кутты первого порядка. Имеет погрешность на шаге и погрешность в целом .

Таблица Бутчера для метода:



Листинг метода:

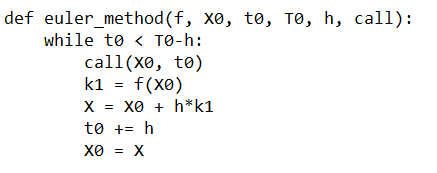


График аналитического решения уравнения и решения методом Эйлера с шагом и представлен на рис. 3, метод вычислил значение функции соответственно на .

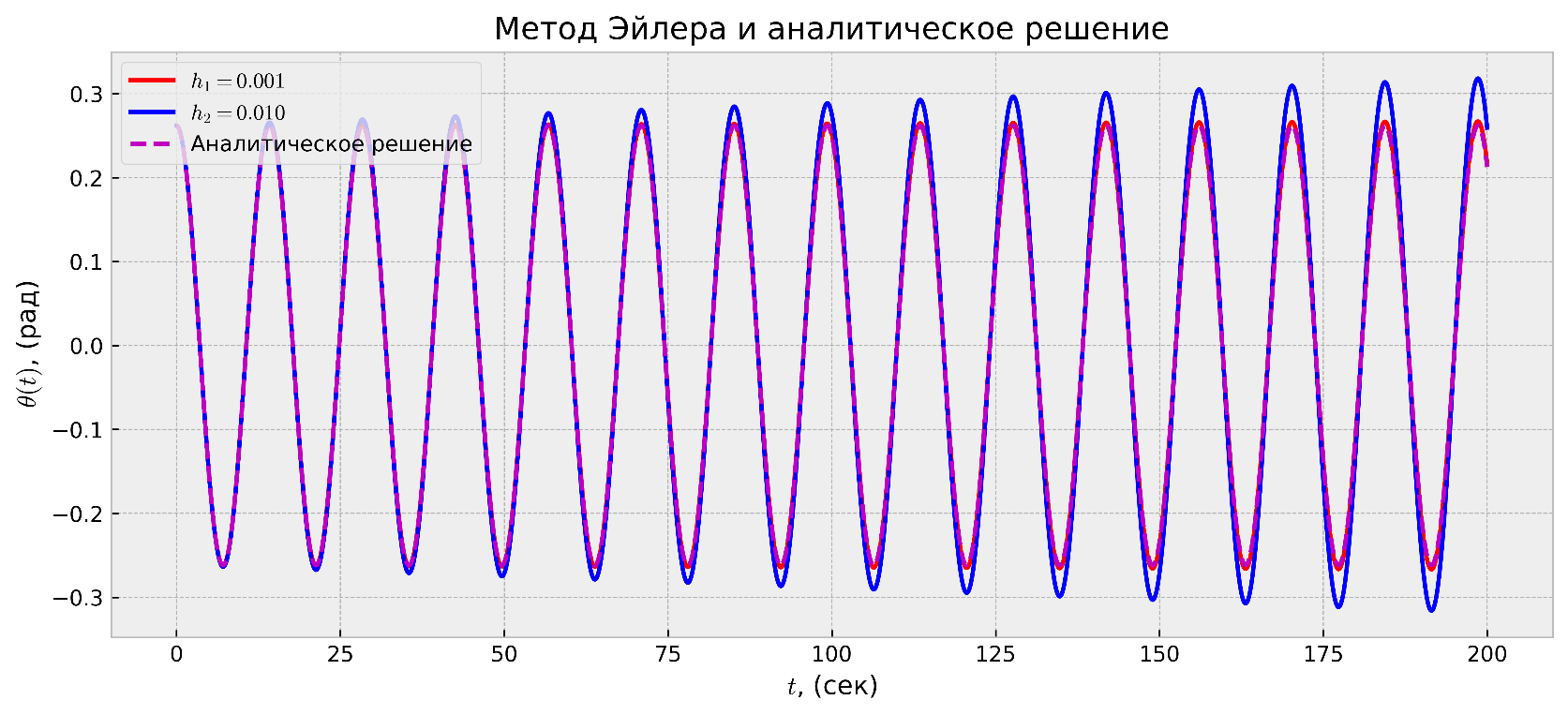


Рисунок 3 – Метод Эйлера и аналитическое решение

График модуля ошибки аналитического решения уравнения и решения методом Эйлера с шагом на :

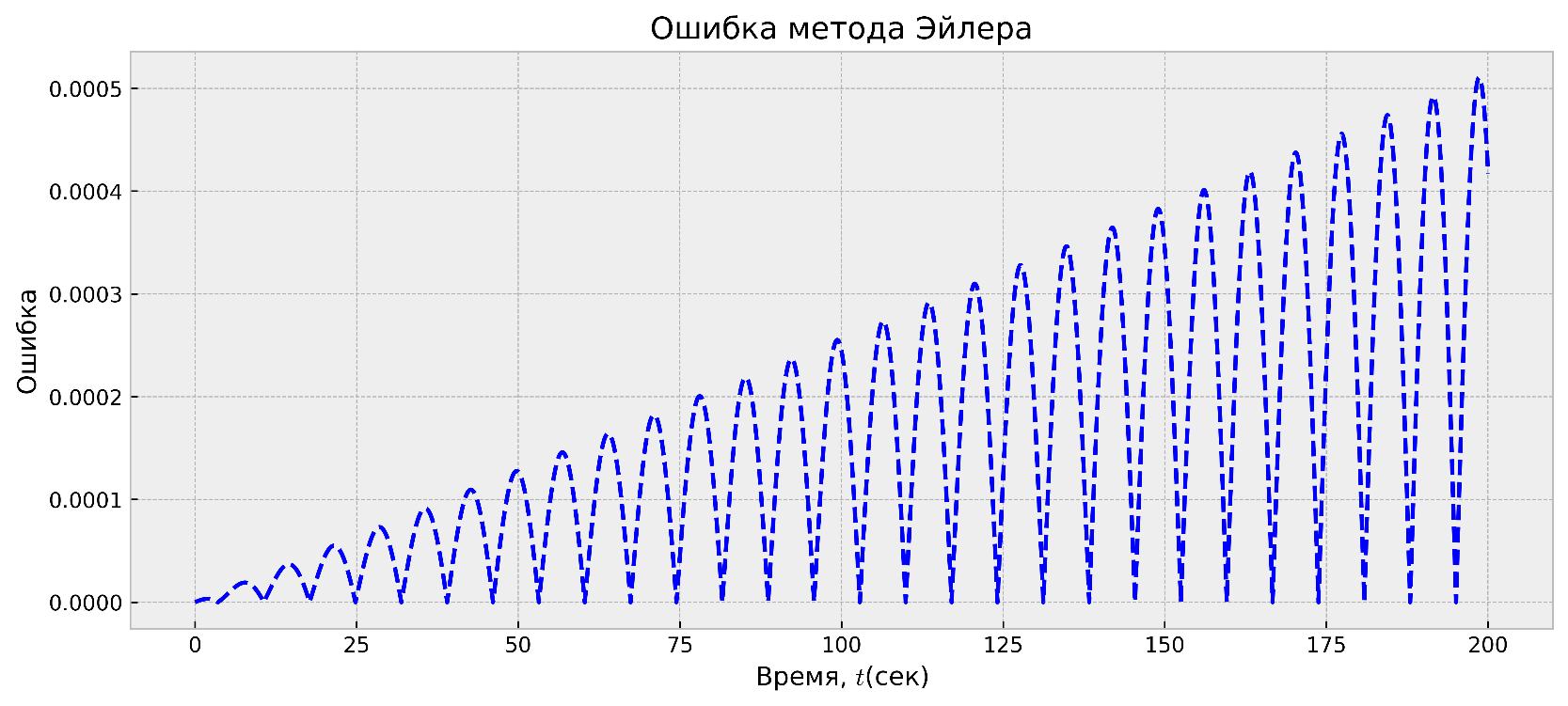


Рисунок 4 – Ошибка метода Эйлера

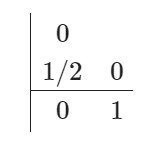
Максимальная ошибка равна ***0.00051***

Видно, что ошибка большая при маленьком шаге, и, 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 и погрешность в целом .

Таблица Бутчера для метода:



Листинг метода:

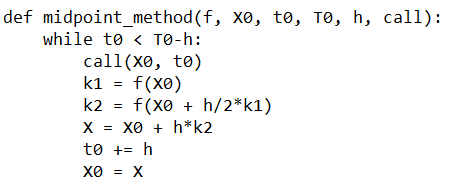


График аналитического решения уравнения и решения методом средней точки с шагом и представлен на рис. 5, метод вычислил значение функции соответственно на

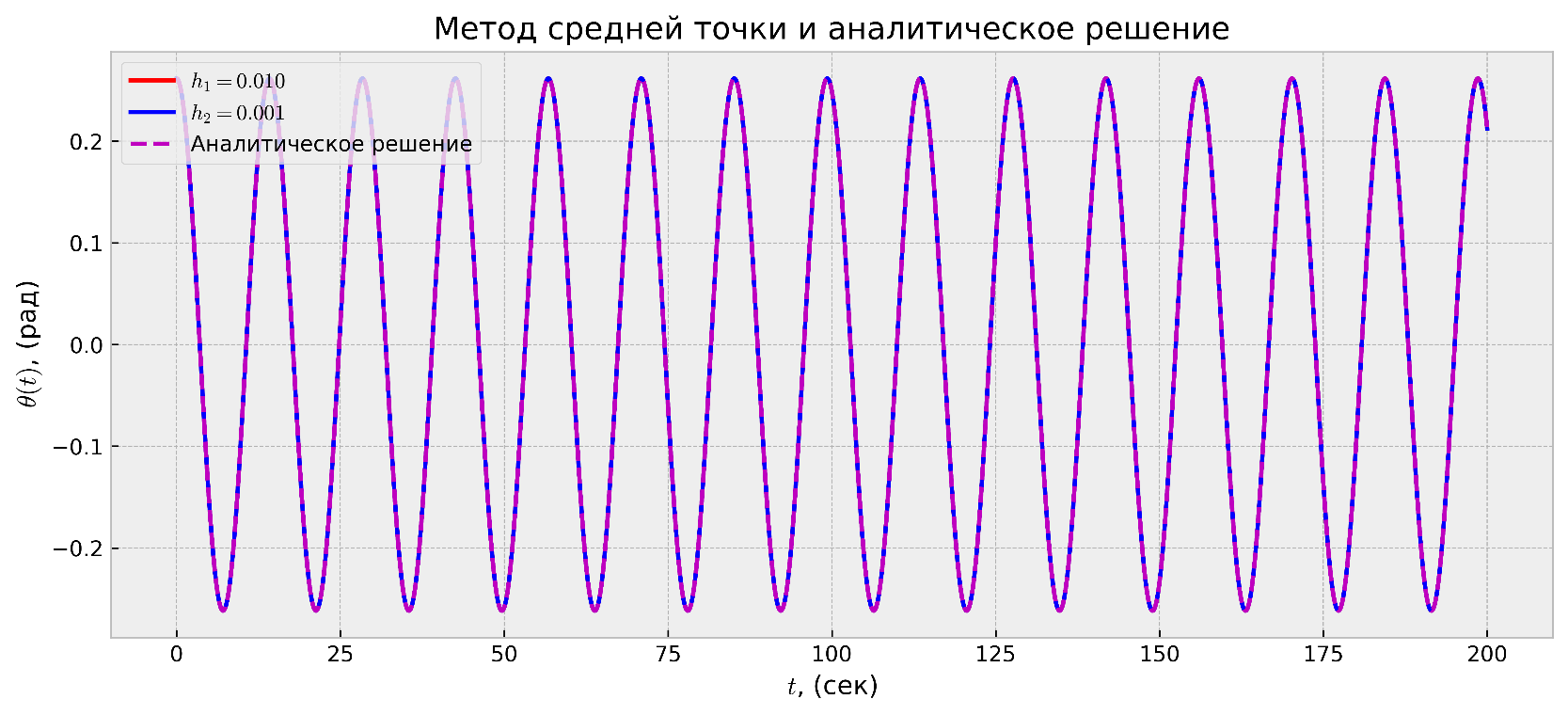


Рисунок 5 – Метод средней точки и аналитическое решение

График модуля ошибки аналитического решения уравнения и решения методом средней точки с шагом на :

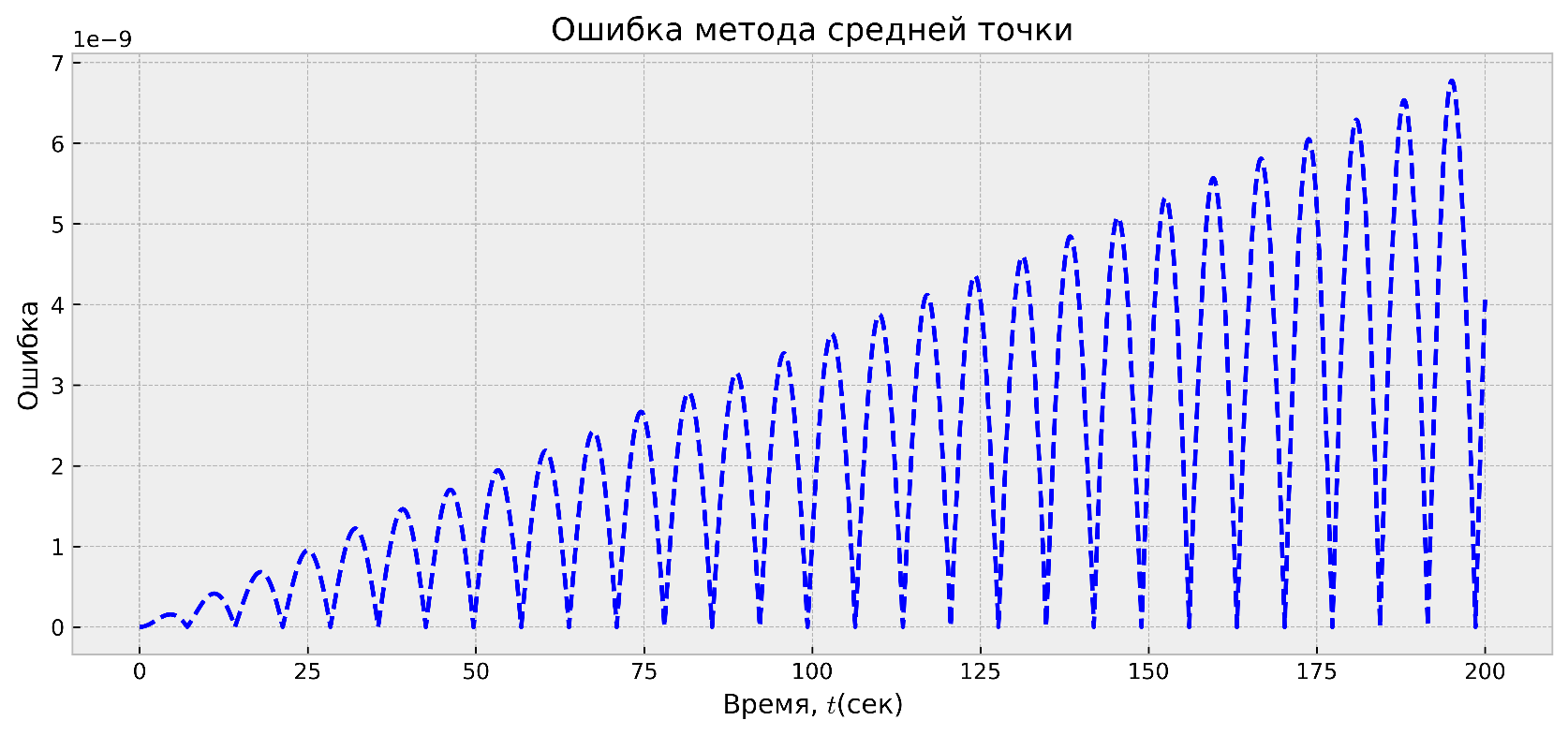


Рисунок 6 – Ошибка метода средней точки

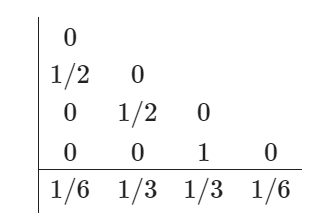
Максимальная ошибка равна ***6.776x10-09***

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.

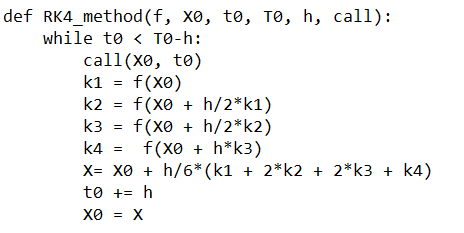
* Метод Рунге-Кутты (4)

Явный одношаговый метод семейства Рунге-Кутты второго порядка. Имеет погрешность на шаге и погрешность в целом .

Таблица Бутчера для метода:



Листинг метода:



Метод вычислил значение функции на

График модуля ошибки аналитического решения уравнения и решения методом Рунге-Кутты 4 с шагом :

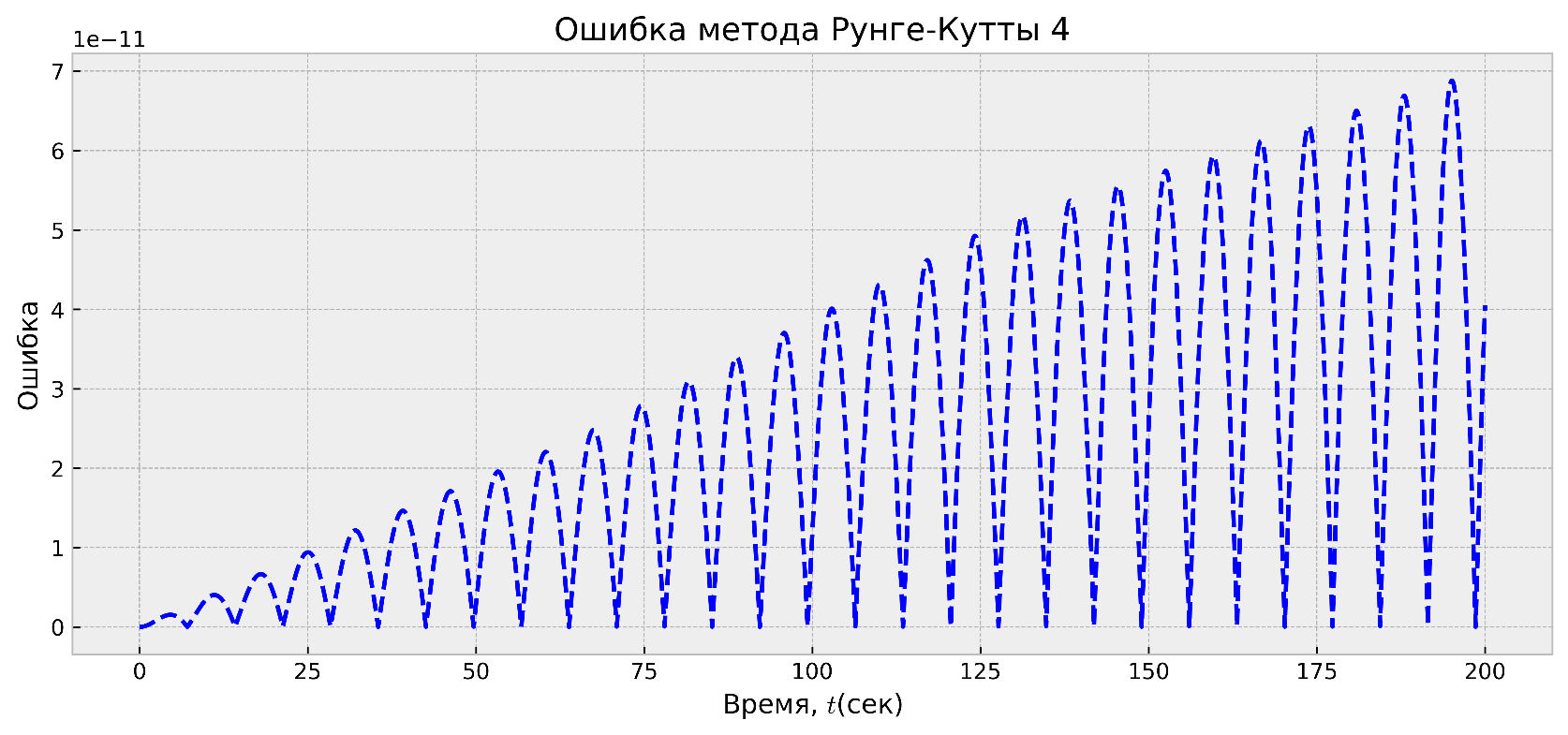


Рисунок 7 – Ошибка метода Рунге-Кутты 4

Максимальная ошибка равна ***6.88x10-11***

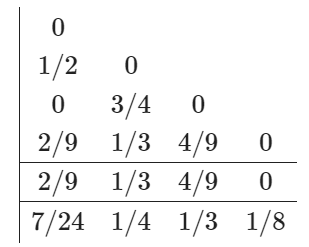
В отличие от предыдущих методов более низких порядков данный метод даже на относительно крупном шаге выдает высокую точность.

***Явные одношаговые вложенные методы***

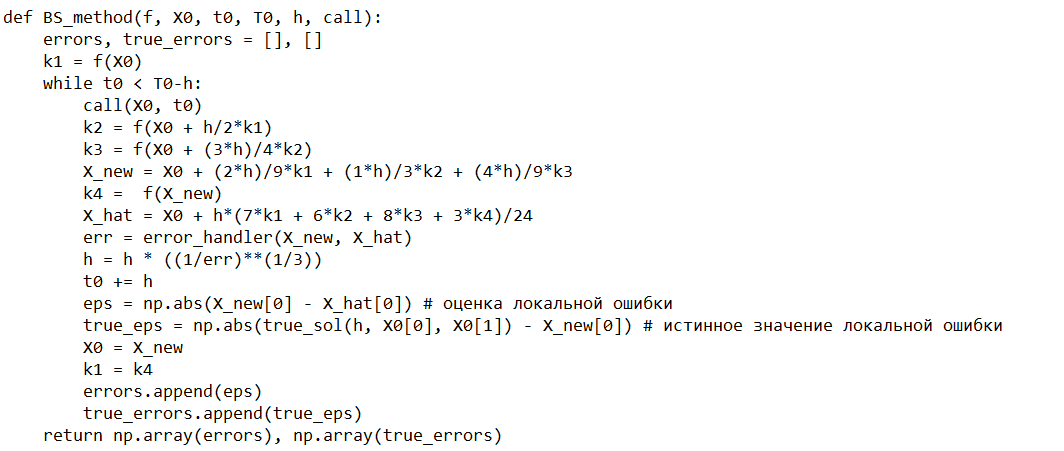
* Метод Богацкого-Шампина (3[2])

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:



Листинг метода:



Метод вычислил значение функции на

На рис. 8 представлен график оценки локальной ошибки метода и истинного значения локальной ошибки. Оценка вычисляется как разность , а истинная локальная ошибка была получена как разность полученного в методе значения и “точного” значения , рассчитанного аналитически с начальным условием .

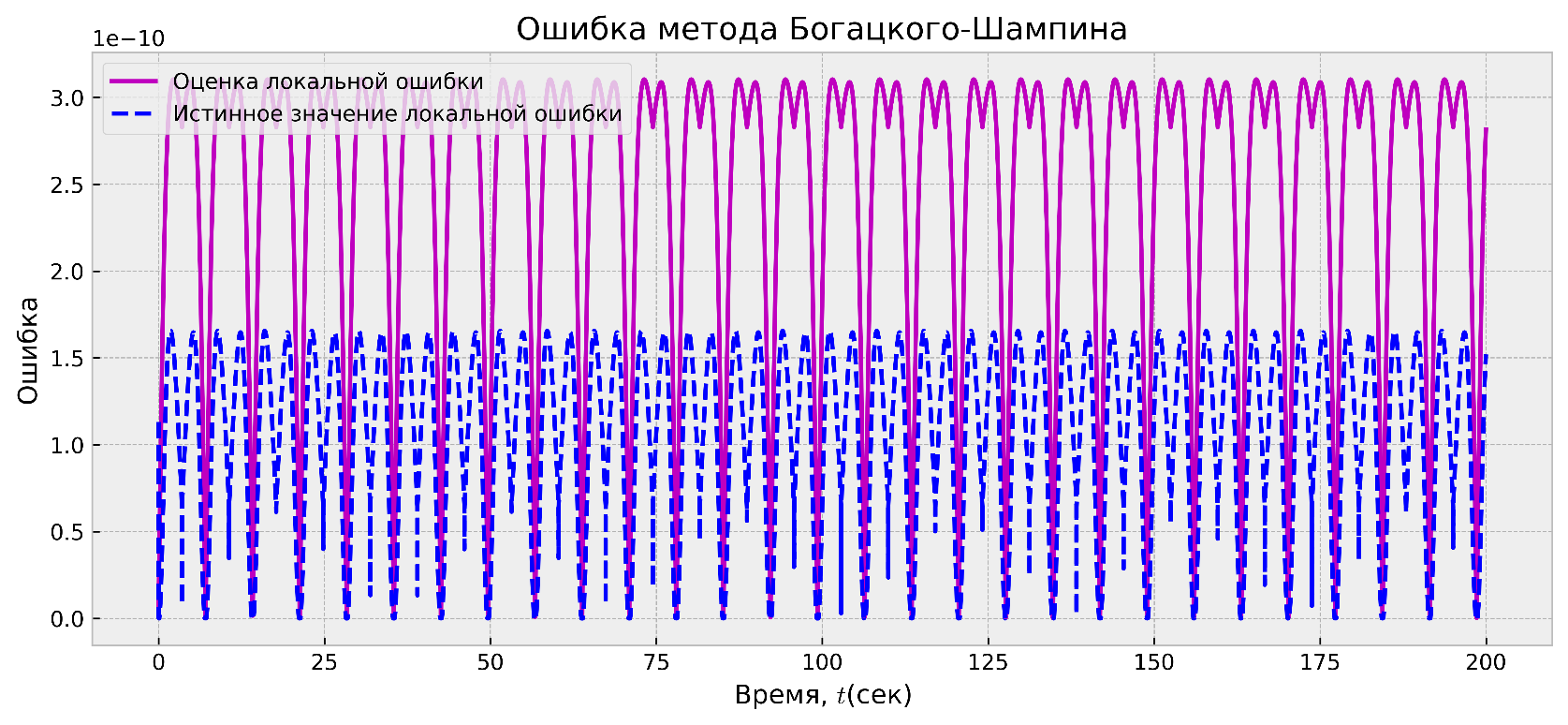


Рисунок 8 – Ошибка метода Богацкого-Шампина

Благодаря адаптивному подбору шага данный метод достигает высокой точности.

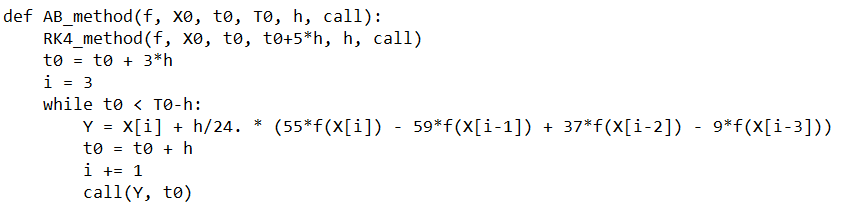
***Многошаговые методы***

* Метод Адамса-Башфорта

Формула для вычисления выглядит следующим образом:

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

Листинг метода:



Метод вычислил значение функции

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

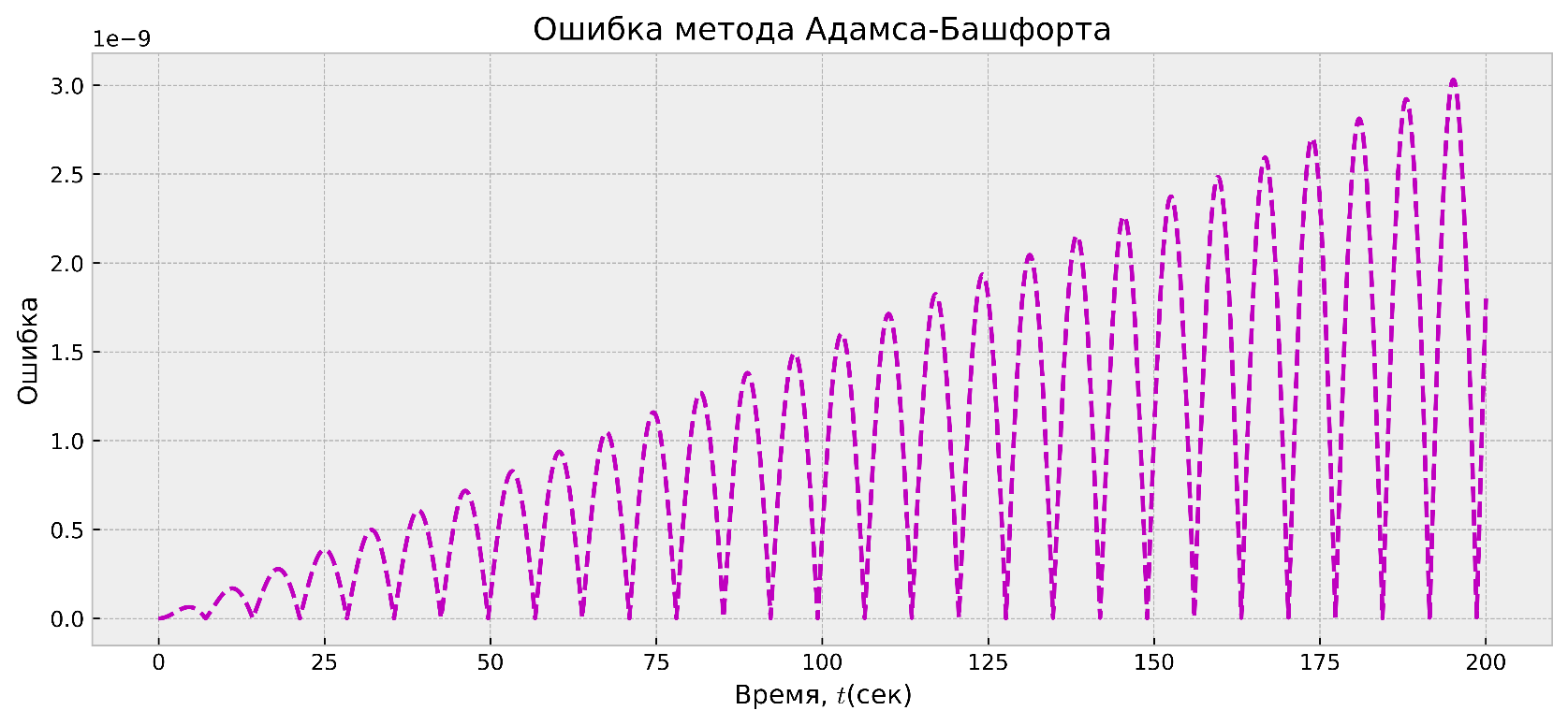
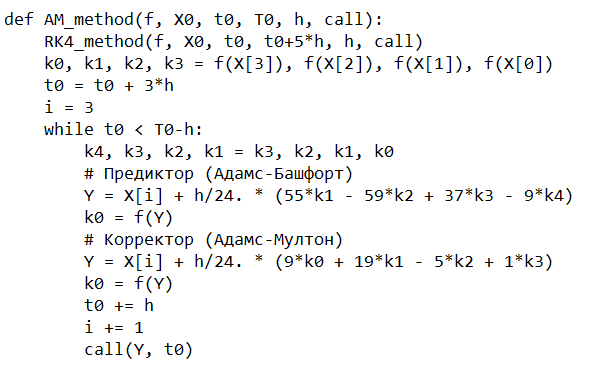


Рисунок 9 – Ошибка метода Адамса-Башфорта

Максимальная ошибка равна ***3.031x10-09***

* Метод Адамса-Мултона

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



Метод вычислил значение функции

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

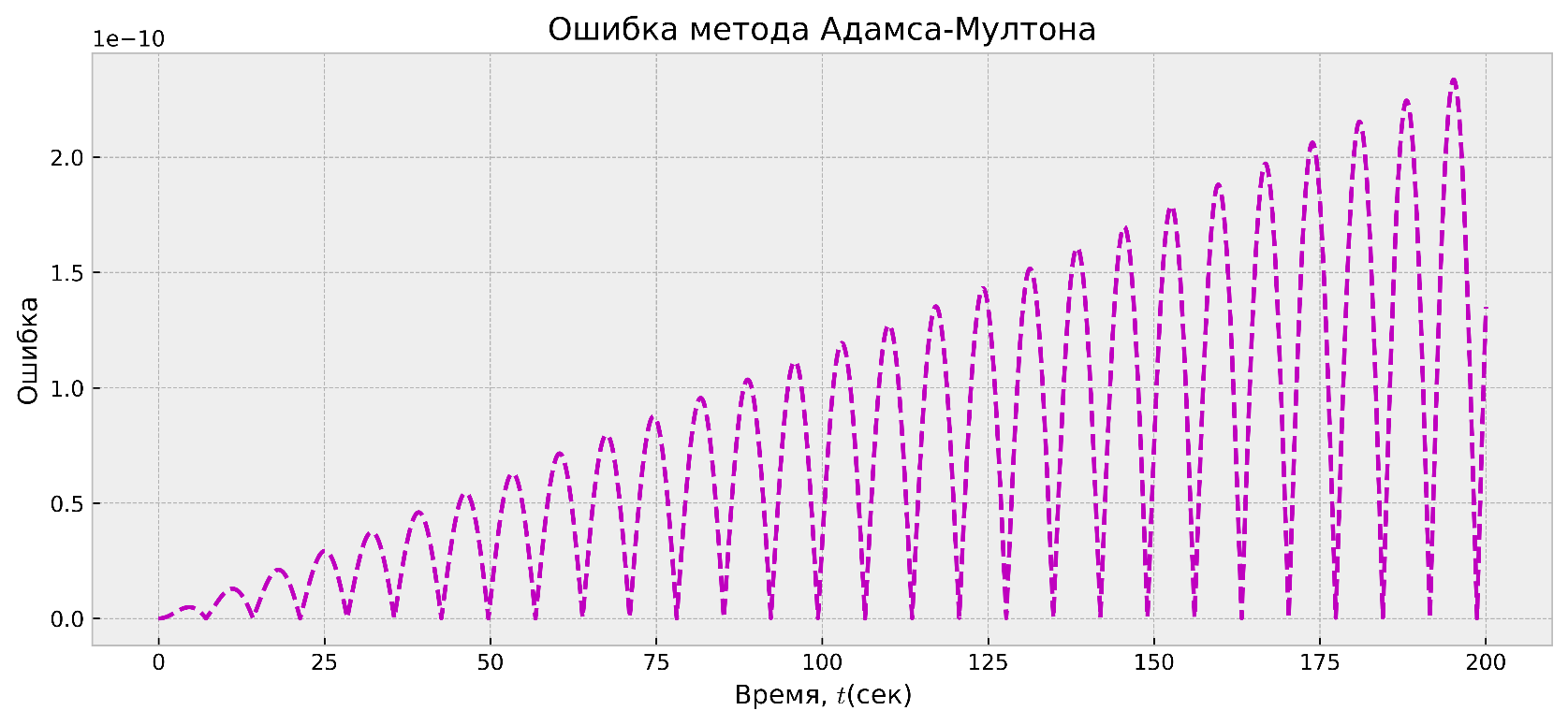


Рисунок 10 – Ошибка метода Адамса-Мултона

Максимальная ошибка равна ***2.336x10-10***

***Сравнение аналитического решения и scipy.odeint***

Было выполнено сравнение аналитического решения задачи и решения, полученного с помощью scipy.odeint на Метод scipy.odeint вычислил значение функции

График модуля ошибки аналитического решения уравнения и решения scipy.odeint :

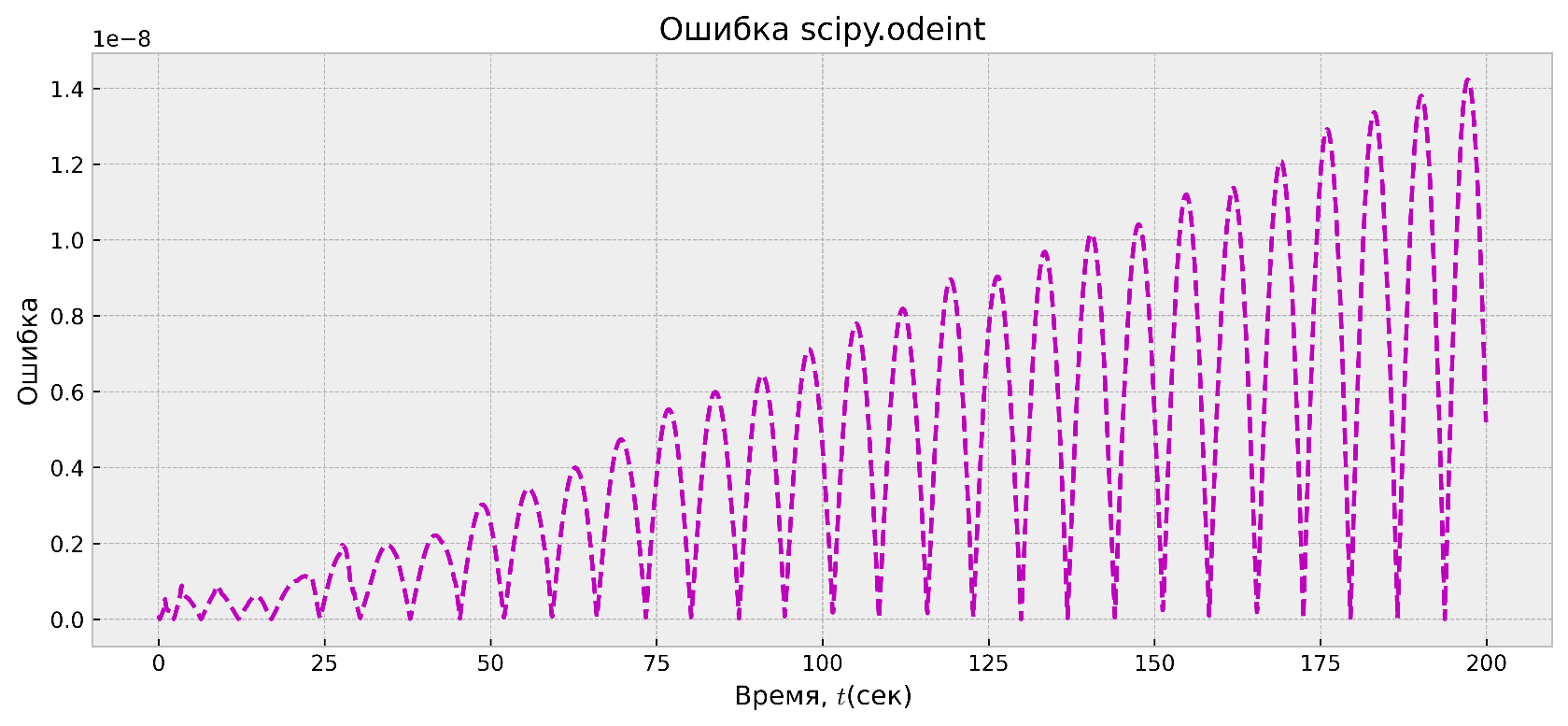


Рисунок 12 – Ошибка scipy.odeint

Максимальная ошибка равна ***1.422x10-08***

***Сравнение методов***

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

Решим задачу с точностью , то есть так, чтобы абсолютная глобальная ошибка была не больше и определим какое количество вызовов функции необходимо методам. Сравнение проводилось на временном промежутке

|  |  |  |
| --- | --- | --- |
| ***Метод*** | ***Шаг*** | ***Количество вызовов функции f*** |
| *Эйлера* | *Очень маленький* | *>1000000* |
| *Средней точки* | *5.075\*10-05* | *788132* |
| *Рунге-Кутты 4* | *0.02288* | *3496* |
| *Богацкого-Шампина* | *Адаптивный шаг* | *6259* |
| *Адамса-Башфорта* | *0.00886* | *8124* |
| *Адамса-Мултона* | *0.01667* | *2412* |

Метод Эйлера проигрывает любому из методов, так как количество вызовов функции 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 для заданной точности , что делает его предпочтительным для решения данной задачи. Также хорошие результаты показал метод Рунге-Кутты 4 порядка, который вызвал функцию 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 на временном промежутке

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

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 необходимо методу Адамса-Башфорта, чтобы не наблюдалось эффекта численного насыщения.

|  |  |  |
| --- | --- | --- |
| ***Шаг*** | ***Количество вызовов f*** | ***Максимальная ошибка*** |
| *0.000786* | *101844* | *4.9741\*10-13* |
| *0.000707* | *113160* | *1.2230\*10-12* |

Видно, что численное насыщение начинает проявляться при количестве вызовов функции

* Проделаем те же действия для метода Адамса-Мултона. Сравним метод с длинами шага на временном промежутке

|  |  |  |
| --- | --- | --- |
| ***Шаг*** | ***Количество вызовов f*** | ***Максимальная ошибка*** |
| *0.1* | *412* | *2.1656\*10-07* |
| *0.01* | *4012* | *2.1081\*10-11* |
| *0.001* | *40012* | *2.8982\*10-13* |
| *0.0001* | *400014* | *3.3037\*10-12* |

Опять же можно заметить, что наблюдается численное насыщение.

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

|  |  |  |
| --- | --- | --- |
| ***Шаг*** | ***Количество вызовов f*** | ***Максимальная ошибка*** |
| *0.000786* | *50934* | *4.8803\*10-13* |
| *0.000707* | *56592* | *1.2164\*10-12* |

Видно, что численное насыщение начинает проявляться при количестве вызовов функции

**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 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)}')