Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВАЯ РАБОТА**

**Решение уравнения Матьё**

по дисциплине «Вычислительная математика»

Выполнил

студент гр. 3530901/10001 <*подпись*> Д.Л.Симоновский

Руководитель

доцент, к.т.н. <*подпись*> В.Н.Цыган

«20» мая 2023 г.

Санкт-Петербург

2023

Санкт-Петербургский государственный политехнический университет

**ЗАДАНИЕ**

**НА ВЫПолнение курсовой работы**

студенту группы 3530901/10001 Симоновский Даниил Леонидович

*(номер группы) (фамилия, имя, отчество)*

***1. Тема работы:*** Решение уравнения Матьё

***2. Срок сдачи законченной работы*** 08июня 2023 г.

***3. Исходные данные к работе*:** Общий вид дифференциального уравнения Матьё; начальные условия для решения; рекомендованный временной интервал исследования численного решения дифференциального уравнения; перечень заданных преподавателем параметров уравнения с указанием уравнений или соотношений для их нахождения (вариант К-3-21).

***4. Содержание пояснительной записки*** (перечень подлежащих разработке вопросов): введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

***Дата получения задания***: «03» марта 2023 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Цыган

*(подпись) (инициалы, фамилия)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Л. Симоновский

*(подпись студента) (инициалы, фамилия)*

\_\_\_\_\_\_\_\_\_\_\_\_

*(дата)*

**Содержание**

[1. Цель работы 3](#_Toc135436428)

[2. Уравнения Матьё 3](#_Toc135436429)

[2.1. История 3](#_Toc135436432)

[2.2. Применение 3](#_Toc135436433)

[3. Ход работы 3](#_Toc135436434)

[3.1. Анализ задания 3](#_Toc135436436)

[3.2. План решения 4](#_Toc135436437)

[3.3. Инструменты 4](#_Toc135436438)

[3.4. Аналитические выкладки 4](#_Toc135436439)

[3.5. Программирование решения 5](#_Toc135436440)

[3.5.1. Система дифференциальных уравнений первого порядка 5](#_Toc135436441)

[3.5.2. Поиск коэффициента A 5](#_Toc135436442)

[3.5.3. Поиск коэффициента E 6](#_Toc135436443)

[3.5.4. Решение системы дифференциальных уравнений 1 порядка 6](#_Toc135436444)

[3.5.5. Оценка погрешности 7](#_Toc135436445)

[3.5.6. Результат работы оценки погрешности 8](#_Toc135436446)

[3.5.7. Оценка влияния погрешности исходных данных 9](#_Toc135436447)

[3.5.8. Результат работы оценки влияния погрешности исходных данных 11](#_Toc135436448)

[4. Анализ результатов 14](#_Toc135436449)

[4.1. Анализ погрешности 14](#_Toc135436451)

[4.2. Анализ устойчивости 14](#_Toc135436452)

[5. Заключение 14](#_Toc135436453)

[6. Источники 14](#_Toc135436454)

[7. Приложение 14](#_Toc135436455)

# Цель работы

Решить уравнение Матьё (1.1)

(1.1)

для следующих значений :

B = 0

= 1

Построить график U(t), очень погрешность результата и влияние на точность погрешности исходных данных.

# Уравнения Матьё



## История

Уравнение Матье, классическое уравнение теории нелинейных колебаний, было введено французским математиком Эмилем Леонардом Матьё (1835-1890 г.) в 1868 году в ходе его исследований, связанных с колебаниями эллиптической мембраны барабана [1]. Уравнение отличается от простого гармонического осциллятора тем, что оно добавляет периодический (изменяющийся во времени) коэффициент жесткости [2].

## Применение

Математические функции Матьё являются периодическими решениями линейного дифференциального уравнения второго порядка, известного как уравнение Матьё:

Они могут быть классифицированы как четные или нечетные функции, в зависимости от своей симметрии относительно оси координат.

Эти функции имеют широкий спектр применений в различных областях науки и техники. Например, функции Матьё используются в теории дифференциальных уравнений, физике колебаний, теории волн, электродинамике и оптике. Они представляют собой мощный инструмент для описания и анализа различных физических явлений [3].

Важно отметить, что функции Матьё имеют сложную аналитическую форму и требуют специальных методов и алгоритмов для их вычисления и исследования. Благодаря современным компьютерным технологиям и численным методам, эти функции стали более доступными для использования и исследования в научных и инженерных приложениях.

Таким образом, функции Матьё представляют собой важный инструмент для математического моделирования и решения различных физических задач. Их широкий спектр применений и уникальные свойства делают их неотъемлемой частью современной науки и техники.

# Ход работы



## Анализ задания

Для решения уравнения 1.1. необходимо вычислить коэффициенты A и E, остальные заданы точно.

Для вычисления коэффициента A необходимо найти корень уравнения , решить таковое аналитически не представляется возможным, воспользуемся для этого программными средствами.

Для вычисления коэффициента E необходимо вычислить значение интеграла . Посчитать его аналитически не выйдет, поэтому, как и для поиска коэффициента A, будем использовать программные средства.

Так же на данный момент мы обладаем навыками решения систем дифференциальных уравнений, первого порядка, поэтому уравнение 1.1. (уравнение второго порядка) необходимо привести к знакомой системе.

## План решения

Составим план по решению дифференциального уравнения 1.1.

1. Аналитически приведем уравнение 1.1. к системе дифференциальных уравнений первого порядка.
2. Программными методами вычислим значение коэффициента A
3. Программными методами вычислим значение коэффициента E
4. Решим полученную на шаге 1 систему уравнений и оценим точность решения.
5. Решим эту же систему, добавив искусственную погрешность в исходные данные уравнения.

## Инструменты

Для работы был выбран язык программирования Python версии 3.11 по причине удобства его использования для поставленной задачи. Были выбраны следующие библиотеки:

1. NumPy – для большей скорости расчетов и простоты обработки
2. SciPy – для функций расчета значения интеграла, поиска минимума функции, решения системы дифференциальных уравнений.
3. PrettyTable – для красивого вывода таблицы в консоль
4. MatPlotLib – для вывода графиков

## Аналитические выкладки

Приведем дифференциального уравнения второго порядка к системе дифференциальных уравнений первого порядка, для этого воспользуемся алгоритмом, приведенном в книге «Вычислительная математика», авторов С. М. Устинов и В. А. Зимницкий [4].

Для наглядности добавим в уравнение 1.1. нулевой элемент тогда наше уравнение будет выглядеть таким образом:

Первым делом необходимо сделать коэффициент 1 при старшей степени, однако в нашем уравнении он уже равен единице, поэтому пропустим это действие.

Выполним замену переменных: и , получим:

Решение этого уравнение эквивалентно решению системы , где , A – матрица Фробениуса вида: . Таким образом получим систему:

Перепишем её, подставив :

Именно эту систему необходимо будет решить.

## Программирование решения

### Система дифференциальных уравнений первого порядка

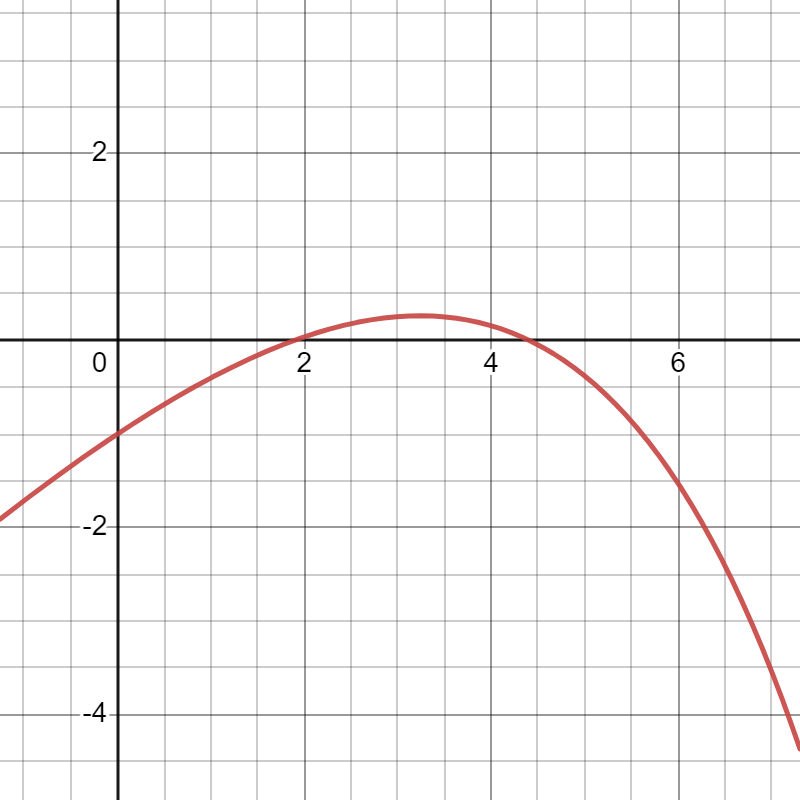
Для дальнейшего использования создадим функцию, которая, основываясь на параметрах и создает и возвращает систему уравнений, полученную в пункте 3.4:

def get\_Mathieu\_function(delt, E):  
 *"""  
 Возвращает функцию Метьё  
 """* def F(t, X):  
 dX = np.zeros(X.shape)  
 dX[1] = X[0]  
 dX[0] = -(delt + E \* np.cos(2 \* t)) \* X[1]  
 return dX  
  
 return F

### Поиск коэффициента A

Для нахождения коэффициента A необходимо умножить число на наименьший корень уравнения .

Перепишем уравнение , перенеся в левую часть и вызвав программу для поиска минимума функции, однако перед этим построим результат в Desmos, чтоб определить начальное приближение (рис. 3.5.1.1).

  
Рис. 3.5.1.1. График функции

Как видим у уравнения 2 корня, около x=2 и x=4. По заданию нам необходим минимальный корень, тогда будем искать ноль функции на промежутке от 0 до 3, начальным приближением возьмем x=2.

Для решения этого уравнения в Python будем использовать метод Ньютона из библиотеки scipy [5]. В качестве параметров ему необходимо передать функцию, ноль которой мы ищем:

def F(x):

return x - (1.4 \*\* x)

Так этот метод требует приближение, которое мы нашли выше и точность:

def get\_x():  
 *"""  
 Возвращает результат решения уравнения x=1.4^x  
 """*   
 x = 2  
 eps = 1e-9  
  
 def F(x):  
 return x - (1.4 \*\* x)  
  
 return scipy.optimize.newton(F, x, tol=eps)

При вызове этого метода мы получаем следующее число: 1.886663306.

Основываясь на графике 3.5.1.1. мы можем с большой уверенностью сказать, что найденный корень является правильным, а также, что мы нашли именно минимальный корень.

### Поиск коэффициента E

Для нахождения коэффициента E необходимо умножить число на значение интеграла .

Для поиска приближенного значения интеграла в Python воспользуемся методом quad библиотеки scipy. Для его использования объявим функцию, интеграл которой собираемся брать:

def F(x):  
 return np.sin(x) / (x \*\* 2 + 1)

Теперь передадим эту функцию в метод quad [6], а также отрезок для интегрирования. Передадим эти параметры в метод и вернем полученное значение, умноженное на 1.553791:

def get\_E():  
 *"""  
 Возвращает переменную E, взяв интеграл и умножив на константу  
 """* start = 0  
 end = 1  
  
 def F(x):  
 return np.sin(x) / (x \*\* 2 + 1)  
  
 return 1.553791 \* scipy.integrate.quad(F, start, end)[0]

### Решение системы дифференциальных уравнений 1 порядка

Для решения системы уравнений первого порядка можно использовать программу rkf45, однако в scipy если более точный метод dop853, который использует явный метод Рунге-Кутты 8 степени [7], им мы и воспользуемся:

def rkf853(f, T, X0):  
 *"""  
 Решает `x' = f(t, x)` для каждого `t` в `T`  
 С начальным значением `X0`, используя явный метод Рунге-Кутта 8  
 """*  
 runge = scipy.integrate.ode(f).set\_integrator('dop853').set\_initial\_value(X0, T[0])  
 X = [X0, \*[runge.integrate(T[i]) for i in range(1, len(T))]]  
 return np.array([i[0] for i in X]), np.array([i[1] for i in X])

Теперь осталось объединить все пункты. Для начала получим значение коэффициентов:

# Вычисляем А на основании x\*  
A = 0.5300355 \* get\_x()  
# B задано  
B = 0  
# Дельта задано  
delt = 1  
# Получим E  
E = get\_E()

Так же получим систему с функцией Матьё:

Mathieu\_function = get\_Mathieu\_function(delt, E)

Теперь осталось задать начальные значения и получить результат вычислений:

points = np.arange(0, 10.5, 0.5)

# Зададим начальные значения  
X0 = np.array([A, B])  
# Получим результат вычисления  
Mathieu = rkf853(Mathieu\_function, points, X0)[0]

Для удобства вынесем весь код по подсчету в отдельную функцию, передавая в неё узлы для счета:

def calculate\_without\_error(points):  
 *"""  
 Вычисляет функцию Матьё  
 """* # Вычисляем А на основании x\*  
 A = 0.5300355 \* get\_x()  
 # B задано  
 B = 0  
 # Дельта задано  
 delt = 1  
 # Получим E  
 E = get\_E()  
 # Получим функцию Матьё  
 Mathieu\_function = get\_Mathieu\_function(delt, E)  
 # Зададим начальные значения  
 X0 = np.array([A, B])  
 # Получим результат вычисления  
 Mathieu = rkf853(Mathieu\_function, points, X0)[0]  
 return Mathieu

Такое название для функции выбрано не просто так, однако об это чуть позже.

### Оценка погрешности

Оценить погрешность напрямую достаточно сложно, что связано с количеством использованных методов и проблемами с необходимостью анализа исходного кода каждого из них для выявления метода, который реально был использован внутри.

Чтоб избежать этого воспользуемся правилом Рунге [8]. Формула для оценки имеет вид:

(3.5.5.1)

p – степень используемого метода. В нашем случае p = 8.

Вызовем метод для подсчета значений функции 2 раза, один раз с шагом 0.5, второй – 0.25:

# Получение решения  
points = np.arange(0, 10.5, 0.5)  
without\_error = calculate\_without\_error(points)  
# Получение решения с двойной точностью  
doubled\_points = np.arange(0, 10.25, 0.25)  
without\_error\_doubled\_points = calculate\_without\_error(doubled\_points)

Используя библиотеку pyplot выведем полученные функции на экран:

# Вывод на график  
plt.plot(points, without\_error, '-o', color='g', label='Step 0.5')  
plt.plot(doubled\_points, without\_error\_doubled\_points, '-o', color='b', label='Step 0.25', alpha=0.3)  
plt.legend()  
plt.savefig('graphs/graphs\_without\_error.png')  
plt.show()

Теперь выполним оценку погрешности по формуле 3.5.5.1 и выведем её на график:

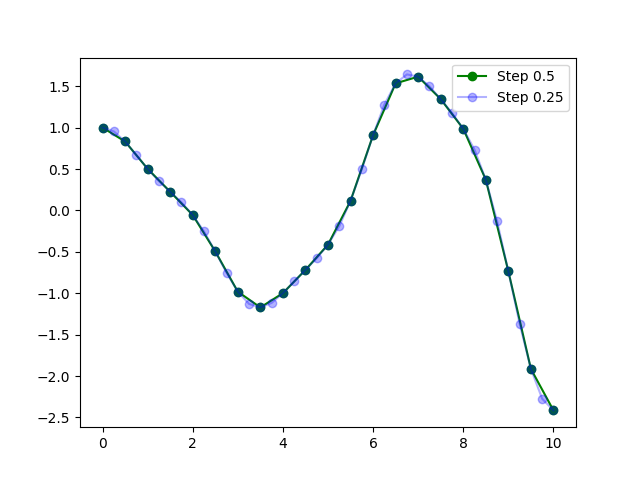
# Оценка точности  
Runge\_rule = abs(without\_error - without\_error\_doubled\_points[::2]) / (2 \*\* 8 - 1)  
plt.plot(points, Runge\_rule, '-o')  
plt.savefig('graphs/error\_of\_method.png')  
plt.show()

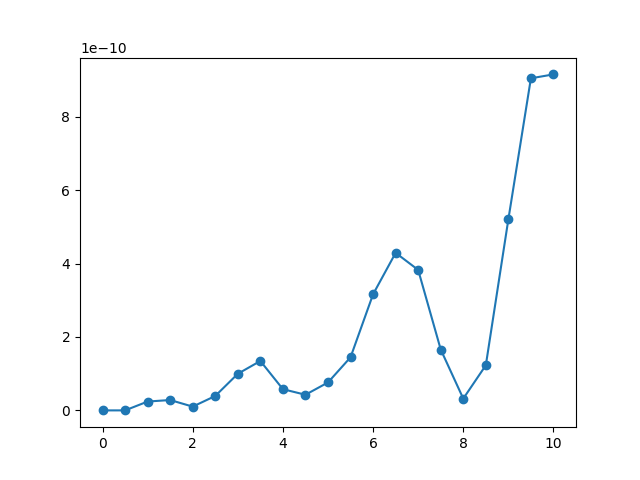
Для удобства выведем её так же в виде таблицы, используя библиотеку PrettyTable:

pt = PrettyTable()  
pt.add\_column('X', points)  
pt.add\_column('Without Error', without\_error)  
pt.add\_column('Without Error Doubled', without\_error\_doubled\_points[::2])  
pt.add\_column('Runge Rule', Runge\_rule)  
print(pt)  
print(f'Max error = {max(Runge\_rule)}')

### Результат работы оценки погрешности

В результате работы кода пункта 3.5.5. и выше мы получаем 2 графика:

  
Рис. 3.5.6.1. Графики U(t) с шагом 0.5 и 0.25

  
Рис. 3.5.6.2. График погрешности по методу Рунге

Так же получаем одну таблицу, в которой идет сравнение графиков с шагом 0.5 и 0.25:

+------+----------------------+-----------------------+------------------------+

| X | Without Error | Without Error Doubled | Runge Rule |

+------+----------------------+-----------------------+------------------------+

| 0.0 | 0.9999985288579282 | 0.9999985288579282 | 0.0 |

| 0.5 | 0.8322980186963664 | 0.8322980186867972 | 3.752640899548686e-14 |

| 1.0 | 0.5020991585507858 | 0.5020991523945542 | 2.4142084845407807e-11 |

| 1.5 | 0.22533999377558472 | 0.22533998657888893 | 2.822233645040498e-11 |

| 2.0 | -0.05559893164245444 | -0.055598934264595806 | 1.028290731431172e-11 |

| 2.5 | -0.4950896274000326 | -0.4950896374212451 | 3.929887265281409e-11 |

| 3.0 | -0.9810392049605126 | -0.9810392303879802 | 9.971555955673341e-11 |

| 3.5 | -1.171384340663171 | -1.1713843749855828 | 1.3459769333948378e-10 |

| 4.0 | -0.998984962381676 | -0.9989849771187479 | 5.779243880840233e-11 |

| 4.5 | -0.7181256913722642 | -0.7181256805090818 | 4.2600715547772934e-11 |

| 5.0 | -0.41290931403308245 | -0.41290929462380604 | 7.611480944435654e-11 |

| 5.5 | 0.11553022687227327 | 0.11553026375978742 | 1.4465691825221837e-10 |

| 6.0 | 0.9085642356395505 | 0.9085643163013983 | 3.1632097203020535e-10 |

| 6.5 | 1.534340255175179 | 1.5343403645884244 | 4.2907155040290497e-10 |

| 7.0 | 1.61494605039075 | 1.6149461479453024 | 3.825668722988533e-10 |

| 7.5 | 1.3437082568084524 | 1.3437082990949099 | 1.6582924488921426e-10 |

| 8.0 | 0.9859012242009985 | 0.9859012323798511 | 3.207393174582929e-11 |

| 8.5 | 0.37059095923155216 | 0.37059092790467335 | 1.2285050514997092e-10 |

| 9.0 | -0.7349853189840385 | -0.7349854516933727 | 5.204287615236032e-10 |

| 9.5 | -1.913642309491197 | -1.9136425401261068 | 9.044506268112935e-10 |

| 10.0 | -2.4145272477430137 | -2.4145274810769797 | 9.150351604965439e-10 |

+------+----------------------+-----------------------+------------------------+

Max error = 9.150351604965439e-10

Табл. 3.5.6.3. Сравнение графиков с шагом 0.5 и 0.25

### Оценка влияния погрешности исходных данных

Для оценки влияния погрешности исходных данных будем увеличивать/уменьшать значение в n – ом знаке исходных данных. Для этого создадим функцию calculate, которая будет работать аналогично функции calculate\_without\_error, однако будет принимать дополнительный параметр error, который будет отвечать за погрешность.

Для корректности эксперимента будем случайным образом менять знак погрешности, используя библиотеку random:

def calculate(points, error):  
 *"""  
 Вычисляет функцию Матьё с заданной погрешностью  
 """* # Вычисляем А на основании x\*  
 A = 0.5300355 \* get\_x() + error \* random.choice([-1, 1])  
 # B задано  
 B = 0 + error \* random.choice([-1, 1])  
 # Дельта задано  
 delt = 1 + error \* random.choice([-1, 1])  
 # Получим E  
 E = get\_E() + error \* random.choice([-1, 1])  
 # Получим функцию Матьё  
 Mathieu\_function = get\_Mathieu\_function(delt, E)  
 # Зададим начальные значения  
 X0 = np.array([A, B])  
 # Получим результат вычисления  
 Mathieu = rkf853(Mathieu\_function, points, X0)[0]

Так же для удобства будем сразу же выводить полученный график:

# Вывели результат в виде графика  
plt.title(f'Error = {error}')  
plt.plot(points, Mathieu, '-o')  
plt.savefig(f'graphs/error\_{error}.png')  
plt.show()  
return Mathieu

Создадим массив с коэффициентами для изменения начальных значений. Для эксперимента будем менять значения сначала в 6, потом в 5 и так до 1 знака:

error\_add = np.array([10 \*\* (-i) for i in range(6, 0, -1)])

Далее вызовем функцию calculate, передавая в неё параметры погрешности:

error = []  
for i in error\_add:  
 error.append(calculate(points, i))

Для анализа полученных данных выведем полученные значения в виде таблицы, используя библиотеку PrettyTable:

# Вывод значений  
pt = PrettyTable()  
pt.add\_column('X', points)  
pt.add\_column('0', [f'{i:.07}' for i in without\_error])  
for error\_val, result in zip(error\_add, error):  
 pt.add\_column(f'{error\_val:.0e}', [f'{i:.07}' for i in result])  
print(pt)

Так же выведем разницу между тем, что мы получили, вызывая функцию без дополнительной погрешности и тем, что получили, добавив погрешность, используя туже библиотеку:

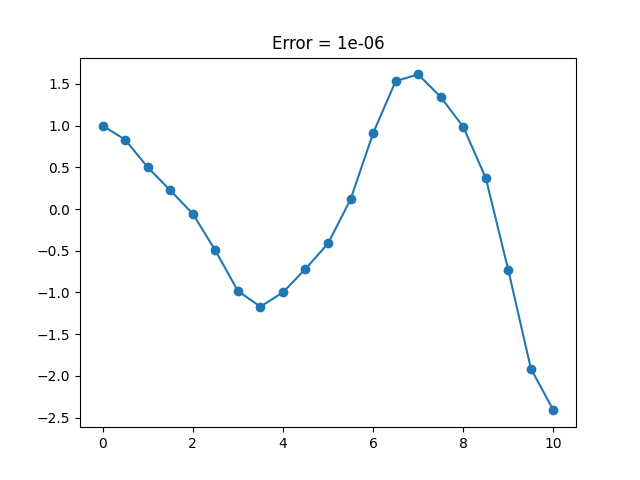
# Вывод погрешностей  
max\_delta = []  
pt = PrettyTable()  
pt.add\_column('X', points)  
for error\_val, result in zip(error\_add, error):  
 max\_delta.append(max(without\_error - result))  
 pt.add\_column(f'{error\_val:.0e}', [f'{i:.05e}' for i in without\_error - result])  
print(pt)

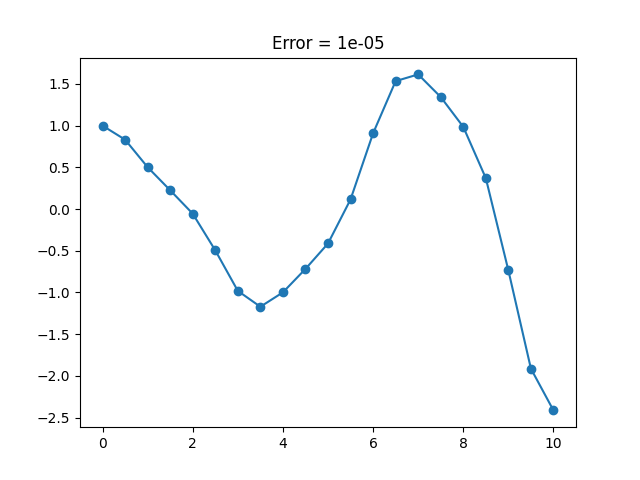
Так же выведем максимальные значения отклонений:

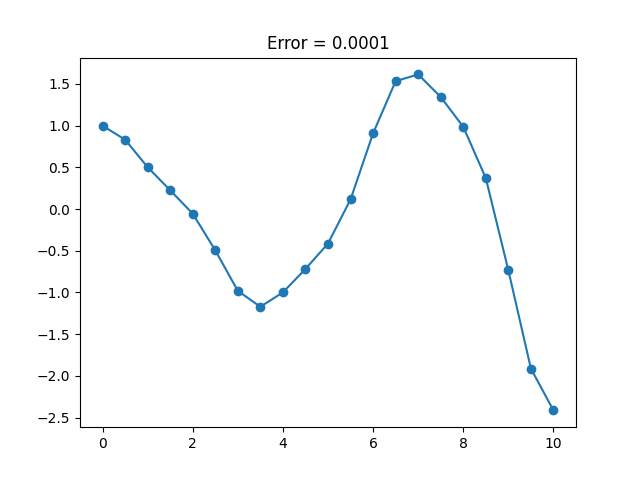
# Вывод максимальных погрешностей  
pt = PrettyTable()  
pt.add\_column('Error val', [f'{i:.0e}' for i in error\_add])  
pt.add\_column('Max Error', [f'{i:.5e}' for i in max\_delta])  
print(pt)

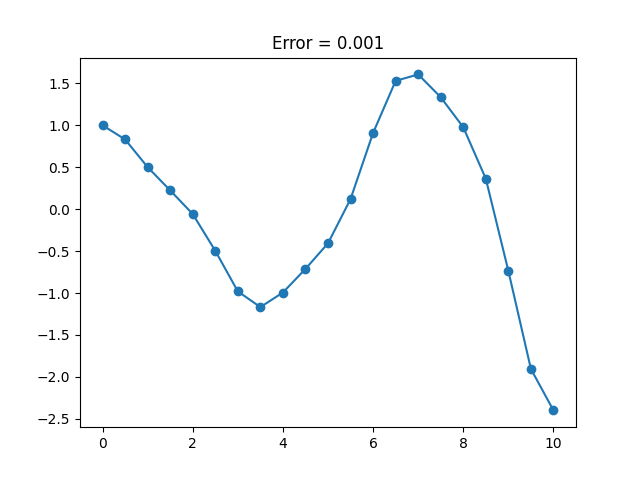
### Результат работы оценки влияния погрешности исходных данных

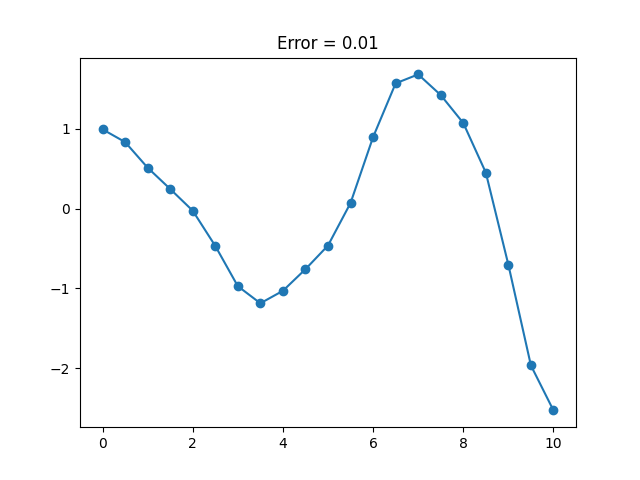
В результате работы кода из пункта 3.5.7. и выше мы получаем 6 графиков (которые при каждом запуске немного отличаются в следствии добавления случайных параметров):

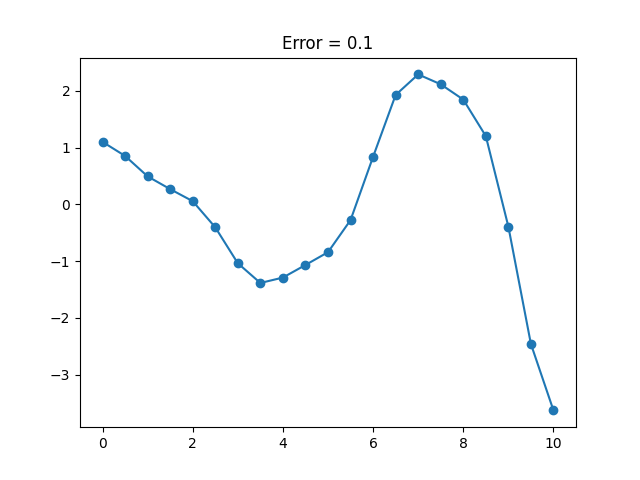
  
Рис. 3.5.8.1. График с погрешностью на 6 знаке

  
Рис. 3.5.8.2. График с погрешностью на 5 знаке

  
Рис. 3.5.8.3. График с погрешностью на 4 знаке

  
Рис. 3.5.8.4. График с погрешностью на 3 знаке

  
Рис. 3.5.8.5. График с погрешностью на 2 знаке

  
Рис. 3.5.8.6. График с погрешностью на 1 знаке

Так же в результате работы мы получаем таблицу по каждому из графиков:

+------+-------------+-------------+-------------+-------------+-------------+-------------+------------+

| X | 0 | 1e-06 | 1e-05 | 1e-04 | 1e-03 | 1e-02 | 1e-01 |

+------+-------------+-------------+-------------+-------------+-------------+-------------+------------+

| 0.0 | 0.9999985 | 0.9999975 | 0.9999885 | 0.9998985 | 0.9989985 | 0.9899985 | 1.099999 |

| 0.5 | 0.832298 | 0.8322981 | 0.8322827 | 0.8323025 | 0.8307683 | 0.8309392 | 0.8523798 |

| 1.0 | 0.5020992 | 0.5021001 | 0.5020808 | 0.5021974 | 0.5002651 | 0.5103346 | 0.4924664 |

| 1.5 | 0.22534 | 0.2253413 | 0.2253165 | 0.2254703 | 0.2229926 | 0.2441924 | 0.267892 |

| 2.0 | -0.05559893 | -0.05559757 | -0.05562604 | -0.05546239 | -0.05830722 | -0.02746436 | 0.05584135 |

| 2.5 | -0.4950896 | -0.4950882 | -0.4951069 | -0.4949443 | -0.4968168 | -0.4678209 | -0.4045383 |

| 3.0 | -0.9810392 | -0.9810381 | -0.9810293 | -0.9809244 | -0.9800514 | -0.9711053 | -1.036762 |

| 3.5 | -1.171384 | -1.171384 | -1.171347 | -1.171382 | -1.167646 | -1.185148 | -1.384108 |

| 4.0 | -0.998985 | -0.9989862 | -0.9989345 | -0.9991122 | -0.993946 | -1.029184 | -1.288871 |

| 4.5 | -0.7181257 | -0.7181276 | -0.7180703 | -0.7183155 | -0.7125864 | -0.7589457 | -1.067397 |

| 5.0 | -0.4129093 | -0.4129113 | -0.4128511 | -0.4131108 | -0.4071003 | -0.4627374 | -0.8420185 |

| 5.5 | 0.1155302 | 0.1155281 | 0.1155741 | 0.1153185 | 0.1199048 | 0.06907962 | -0.2770604 |

| 6.0 | 0.9085642 | 0.9085622 | 0.908561 | 0.9083656 | 0.9082293 | 0.8931143 | 0.8375311 |

| 6.5 | 1.53434 | 1.534339 | 1.534278 | 1.534255 | 1.528137 | 1.565173 | 1.926112 |

| 7.0 | 1.614946 | 1.614947 | 1.614851 | 1.615046 | 1.605468 | 1.67709 | 2.287984 |

| 7.5 | 1.343708 | 1.343711 | 1.343606 | 1.343937 | 1.333498 | 1.418737 | 2.113977 |

| 8.0 | 0.9859012 | 0.9859039 | 0.9857981 | 0.9861728 | 0.9756093 | 1.069325 | 1.844639 |

| 8.5 | 0.370591 | 0.370594 | 0.3705058 | 0.3708954 | 0.3620996 | 0.4490747 | 1.202127 |

| 9.0 | -0.7349853 | -0.7349819 | -0.7350013 | -0.7346413 | -0.7365689 | -0.7030034 | -0.3918448 |

| 9.5 | -1.913642 | -1.91364 | -1.913554 | -1.913369 | -1.904841 | -1.962307 | -2.45787 |

| 10.0 | -2.414527 | -2.414527 | -2.414368 | -2.41449 | -2.398649 | -2.524613 | -3.627452 |

+------+-------------+-------------+-------------+-------------+-------------+-------------+------------+

Табл. 3.5.8.1. Значения графиков

И таблицу погрешностей этих графиков:

+------+--------------+--------------+--------------+--------------+--------------+--------------+

| X | 1e-06 | 1e-05 | 1e-04 | 1e-03 | 1e-02 | 1e-01 |

+------+--------------+--------------+--------------+--------------+--------------+--------------+

| 0.0 | 1.00000e-06 | 1.00000e-05 | 1.00000e-04 | 1.00000e-03 | 1.00000e-02 | -1.00000e-01 |

| 0.5 | -4.43008e-08 | 1.52971e-05 | -4.45480e-06 | 1.52973e-03 | 1.35887e-03 | -2.00818e-02 |

| 1.0 | -9.82650e-07 | 1.83410e-05 | -9.82264e-05 | 1.83409e-03 | -8.23541e-03 | 9.63277e-03 |

| 1.5 | -1.30407e-06 | 2.34828e-05 | -1.30338e-04 | 2.34741e-03 | -1.88525e-02 | -4.25520e-02 |

| 2.0 | -1.36622e-06 | 2.71100e-05 | -1.36538e-04 | 2.70828e-03 | -2.81346e-02 | -1.11440e-01 |

| 2.5 | -1.45442e-06 | 1.73098e-05 | -1.45353e-04 | 1.72715e-03 | -2.72687e-02 | -9.05513e-02 |

| 3.0 | -1.14869e-06 | -9.85788e-06 | -1.14810e-04 | -9.87784e-04 | -9.93391e-03 | 5.57225e-02 |

| 3.5 | -2.10253e-08 | -3.73934e-05 | -2.10085e-06 | -3.73805e-03 | 1.37637e-02 | 2.12723e-01 |

| 4.0 | 1.27257e-06 | -5.04211e-05 | 1.27214e-04 | -5.03899e-03 | 3.01987e-02 | 2.89886e-01 |

| 4.5 | 1.89820e-06 | -5.54387e-05 | 1.89761e-04 | -5.53931e-03 | 4.08200e-02 | 3.49271e-01 |

| 5.0 | 2.01503e-06 | -5.81682e-05 | 2.01443e-04 | -5.80904e-03 | 4.98280e-02 | 4.29109e-01 |

| 5.5 | 2.11771e-06 | -4.38513e-05 | 2.11722e-04 | -4.37453e-03 | 4.64506e-02 | 3.92591e-01 |

| 6.0 | 1.98616e-06 | 3.27731e-06 | 1.98610e-04 | 3.34972e-04 | 1.54499e-02 | 7.10332e-02 |

| 6.5 | 8.49719e-07 | 6.20463e-05 | 8.50274e-05 | 6.20307e-03 | -3.08331e-02 | -3.91772e-01 |

| 7.0 | -1.00075e-06 | 9.48637e-05 | -9.99805e-05 | 9.47832e-03 | -6.21436e-02 | -6.73038e-01 |

| 7.5 | -2.28374e-06 | 1.02215e-04 | -2.28279e-04 | 1.02102e-02 | -7.50288e-02 | -7.70268e-01 |

| 8.0 | -2.71620e-06 | 1.03083e-04 | -2.71543e-04 | 1.02919e-02 | -8.34240e-02 | -8.58738e-01 |

| 8.5 | -3.04433e-06 | 8.51342e-05 | -3.04392e-04 | 8.49133e-03 | -7.84837e-02 | -8.31536e-01 |

| 9.0 | -3.44028e-06 | 1.60137e-05 | -3.44066e-04 | 1.58358e-03 | -3.19819e-02 | -3.43141e-01 |

| 9.5 | -2.73189e-06 | -8.80077e-05 | -2.73333e-04 | -8.80095e-03 | 4.86652e-02 | 5.44227e-01 |

| 10.0 | -3.67469e-07 | -1.58946e-04 | -3.69541e-05 | -1.58783e-02 | 1.10085e-01 | 1.21292e+00 |

+------+--------------+--------------+--------------+--------------+--------------+--------------+

Табл. 3.5.8.2. Погрешности графиков, относительно графика без возмущений

А также таблица с максимальными погрешностями:

+-----------+-------------+

| Error val | Max Error |

+-----------+-------------+

| 1e-06 | 2.11771e-06 |

| 1e-05 | 1.03083e-04 |

| 1e-04 | 2.11722e-04 |

| 1e-03 | 1.02919e-02 |

| 1e-02 | 1.10085e-01 |

| 1e-01 | 1.21292e+00 |

+-----------+-------------+

Табл. 3.5.8.3. Максимальные значения погрешностей

# Анализ результатов



## Анализ погрешности

Как видно из рисунка 3.5.6.2. погрешность возрастает со степенью отдаления от начальных условий, что вполне ожидаемо. Так же видно, что погрешность имеет одинаковый порядок, а именно 10-10, что свидетельствует об отличной степени точности выбранного метода.

## Анализ устойчивости

Как видно из таблицы 3.5.8.2. погрешность имеет примерно тот же порядок, что и порядок, в котором мы меняли значение, что свидетельствует о хорошей устойчивости решения.

# Заключение

В ходе курсовой работы было произведено исследование уравнение Матьё. Для исследования этого уравнения оно было сведено к системе дифференциальных уравнений первого порядка, найдены необходимые коэффициенты, используя метод Ньютона и quad, и было решено уравнение, используя метод Рунге-Кутты 8 степени.

Выполнив анализ полученных результатов, стало понятно, что выбранный метод решения дает ответ с хорошей точностью, а также заданные коэффициенты дают хорошую устойчивость решения.

# Источники

1. Gutierrez-Vega, J. et al. (2003). Mathieu functions, a visual approach. из: https://www.academia.edu/4329322/Mathieu\_functions\_a\_visual\_approach
2. Mathieu, E.L. Journal de Mathematiques, Vol. 13 (2), 1868, –137 c.
3. Уиттекер Э.Т., Ватсон Дж.Н. Курс современного анализа, Том 2, 1963 г – 500 с.
4. Устинов С.М., Зимницкий В.А. Вычислительная математика, 2009 г. – 336 с.
5. Документация метода newton, библиотеки scipy: официальный сайт. – URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html>
6. Документация метода quad, библиотеки scipy: официальный сайт. – URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html>
7. Документация метода quad, библиотеки scipy: официальный сайт. – URL: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.DOP853.html>
8. И. М. Виноградов. Математическая энциклопедия. 1977—1985 г

# Приложение

Код на GitHub, URL: <https://github.com/DafterT/comp_math_coursework>

Листинг кода:

import random  
import matplotlib.pyplot as plt  
import numpy as np  
import scipy  
from prettytable import PrettyTable  
  
  
def get\_x():  
 *"""  
 Возвращает результат решения уравнения x=1.4^x  
 """* # Начальное приближение  
 x = 2  
 eps = 1e-9  
  
 # Функция, у которой ищем ноль  
 def F(x):  
 return x - (1.4 \*\* x)  
  
 return scipy.optimize.newton(F, x, tol=eps)  
  
  
def get\_E():  
 *"""  
 Возвращает переменную E, взяв интеграл и умножив на константу  
 """* start = 0  
 end = 1  
  
 # Функция для интегрирования  
 def F(x):  
 return np.sin(x) / (x \*\* 2 + 1)  
  
 return 1.553791 \* scipy.integrate.quad(F, start, end)[0]  
  
  
def get\_Mathieu\_function(delt, E):  
 *"""  
 Возвращает функцию Метьё  
 """* def F(t, X):  
 dX = np.zeros(X.shape)  
 dX[1] = X[0]  
 dX[0] = -(delt + E \* np.cos(2 \* t)) \* X[1]  
 return dX  
  
 return F  
  
  
def rkf853(f, T, X0):  
 *"""  
 Решает `x' = f(t, x)` для каждого `t` в `T`  
 С начальным значением `X0`, используя явный метод Рунге-Кутта 8  
 """* runge = scipy.integrate.ode(f).set\_integrator('dop853').set\_initial\_value(X0, T[0])  
 X = [X0, \*[runge.integrate(T[i]) for i in range(1, len(T))]]  
 return np.array([i[0] for i in X]), np.array([i[1] for i in X])

def calculate(points, error):  
 *"""  
 Вычисляет функцию Матьё с заданной погрешностью  
 """* # Вычисляем А на основании x\*  
 A = 0.5300355 \* get\_x() + error \* random.choice([-1, 1])  
 # B задано  
 B = 0 + error \* random.choice([-1, 1])  
 # Дельта задано  
 delt = 1 + error \* random.choice([-1, 1])  
 # Получим E  
 E = get\_E() + error \* random.choice([-1, 1])  
 # Получим функцию Матьё  
 Mathieu\_function = get\_Mathieu\_function(delt, E)  
 # Зададим начальные значения  
 X0 = np.array([A, B])  
 # Получим результат вычисления  
 Mathieu = rkf853(Mathieu\_function, points, X0)[0]  
 # Вывели результат в виде графика  
 plt.title(f'Error = {error}')  
 plt.plot(points, Mathieu, '-o')  
 plt.savefig(f'graphs/error\_{error}.png')  
 plt.show()  
 return Mathieu  
  
  
def calculate\_without\_error(points):  
 *"""  
 Вычисляет функцию Матьё  
 """* # Вычисляем А на основании x\*  
 A = 0.5300355 \* get\_x()  
 # B задано  
 B = 0  
 # Дельта задано  
 delt = 1  
 # Получим E  
 E = get\_E()  
 # Получим функцию Матьё  
 Mathieu\_function = get\_Mathieu\_function(delt, E)  
 # Зададим начальные значения  
 X0 = np.array([A, B])  
 # Получим результат вычисления  
 Mathieu = rkf853(Mathieu\_function, points, X0)[0]  
 return Mathieu

def main():  
 # Получение решения  
 points = np.arange(0, 10.5, 0.5)  
 without\_error = calculate\_without\_error(points)  
 # Получение решения с двойной точностью  
 doubled\_points = np.arange(0, 10.25, 0.25)  
 without\_error\_doubled\_points = calculate\_without\_error(doubled\_points)  
 # Вывод на график  
 plt.plot(points, without\_error, '-o', color='g', label='Step 0.5')  
 plt.plot(doubled\_points, without\_error\_doubled\_points, '-o', color='b', label='Step 0.25', alpha=0.3)  
 plt.legend()  
 plt.savefig('graphs/graphs\_without\_error.png')  
 plt.show()  
 # Оценка точности  
 Runge\_rule = abs(without\_error - without\_error\_doubled\_points[::2]) / (2 \*\* 8 - 1)  
 plt.plot(points, Runge\_rule, '-o')  
 plt.savefig('graphs/error\_of\_method.png')  
 plt.show()  
 pt = PrettyTable()  
 pt.add\_column('X', points)  
 pt.add\_column('Without Error', without\_error)  
 pt.add\_column('Without Error Doubled', without\_error\_doubled\_points[::2])  
 pt.add\_column('Runge Rule', Runge\_rule)  
 print(pt)  
 print(f'Max error = {max(Runge\_rule)}')  
 # Оценка влияния погрешности исходных данных  
 error\_add = np.array([10 \*\* (-i) for i in range(6, 0, -1)])  
 error = []  
 for i in error\_add:  
 error.append(calculate(points, i))  
 # Вывод значений  
 pt = PrettyTable()  
 pt.add\_column('X', points)  
 pt.add\_column('0', [f'{i:.07}' for i in without\_error])  
 for error\_val, result in zip(error\_add, error):  
 pt.add\_column(f'{error\_val:.0e}', [f'{i:.07}' for i in result])  
 print(pt)  
 # Вывод погрешностей  
 max\_delta = []  
 pt = PrettyTable()  
 pt.add\_column('X', points)  
 for error\_val, result in zip(error\_add, error):  
 max\_delta.append(max(without\_error - result))  
 pt.add\_column(f'{error\_val:.0e}', [f'{i:.05e}' for i in without\_error - result])  
 print(pt)  
 # Вывод максимальных погрешностей  
 pt = PrettyTable()  
 pt.add\_column('Error val', [f'{i:.0e}' for i in error\_add])  
 pt.add\_column('Max Error', [f'{i:.5e}' for i in max\_delta])  
 print(pt)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()