## Национальный исследовательский университет «МЭИ»

## (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №6:

## «ЧИСЛЕННОЕ ДИФФЕРЕНЦИРОВАНИЕ и РЕШЕНИЕ ЗАДАЧИ КОШИ»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 41

2021 Москва

**Постановка задачи:**

**Задача 6.1.** Исследовать поведение погрешностей при численном дифференцировании функции.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:**

1. Взять функцию из задачи 5.2. Выбрать фиксированную точку c на отрезке [a,b] и вычислить значения производных, указанных в индивидуальном варианте в точке с.

2. Задать массив шагов , *k*=1,…15, и вычислить массивы приближенныx значений производных в точке  по формуле (1) и по формуле из индивидуального варианта (для примера взята формула (4)):

 (1) и  (4) ,  *k*=1,…15. Вычислить также массивы значений погрешностей:  и 

3.По полученным таблицам результатов найти оптимальное значение шага дифференцирования для каждого метода. Результаты вычислений внести в первую часть таблицы (см. **ПРИЛОЖЕНИЕ 6.B).**

4. Проделать те же вычисления для производной более высокого порядка, указанной в индивидуальном варианте. Найти оптимальное значение шага дифференцирования, результаты внести во вторую часть таблицы (см. **ПРИЛОЖЕНИЕ 6.B).**

5. По полученным данным построить графики погрешностей

6. Вывести оценку погрешности указанной в индивидуальном варианте формулы.

|  |  |  |  |
| --- | --- | --- | --- |
| 5.2.41 |  | | [0,10] |
| 6,13,20,27,34,**41**,48,55 | | | : центральная разностная производная (3)  : односторонняя левая производная второго порядка (8)  Для четных вариантов вывод формулы (8),для нечетных вариантов вывод формулы (8а) | |

**Теоретический материал:**

Формулы численного дифференцирования **первого порядка точности**

(**1**)  **остаточный** член (**1а**) 

Формулы численного дифференцирования **второго порядка точности**

(**3**) **Центральная** разностная производная

Односторонние формулы численного дифференцирования **второго порядка**:

(**8**) **одностороння левая** производная для вычисления **второй производной** **(8a) остаточный** член 

**Код программы** *(пояснения к коду зеленым цветом)***:**

# %%

import numpy

import math

from prettytable import PrettyTable

from sympy import \*

import matplotlib.pyplot as plt

# функция из задачи 5.2

f = lambda x: math.sqrt(x) \* (math.sin(3 \* x)) \*\* 2

# формула центральной разностной производной

df\_central = lambda x, h: (f(x + h) - f(x - h)) / 2 / h

# формула правой разностной производной

df\_right = lambda x, h: (f(x + h) - f(x)) / h

# формула односторонней левой производной

# для вычисления второй производной

d2f\_left = (

    lambda x, h: (2 \* f(x) - 5 \* f(x - h) + 4 \* f(x - 2 \* h) - f(x - 3 \* h)) / h / h

)

# погрешность вычисления производной

df\_error = lambda df\_value, c, n: abs(df\_value - df(c, n))

# функция возвращает значение производной порядка n в точке t

# производная вычисляется с помощью встроенной библиотеки sympy

def df(t, n):

    x = Symbol("x")

    y = sqrt(x) \* (sin(3 \* x)) \*\* 2

    dx = diff(y, x, n)

    ddx = lambdify(x, dx)

    return ddx(t)

def count\_arrays(c):

    h = []

    df\_right\_values = []

    df\_central\_values = []

    d2f\_left\_values = []

    df\_right\_errors = []

    df\_central\_errors = []

    d2f\_left\_errors = []

    # заполнение всех массивов, указанных в задаче

    for k in range(1, 16):

        h.append(10 \*\* (-k))

        df\_right\_values.append(df\_right(c, h[k - 1]))

        df\_central\_values.append(df\_central(c, h[k - 1]))

        d2f\_left\_values.append(d2f\_left(c, h[k - 1]))

        df\_right\_errors.append(df\_error(df\_right\_values[k - 1], c, 1))

        df\_central\_errors.append(df\_error(df\_central\_values[k - 1], c, 1))

        d2f\_left\_errors.append(df\_error(d2f\_left\_values[k - 1], c, 2))

    table = PrettyTable()

    table.add\_column("h", h)

    table.add\_column("Right derivative", df\_right\_values)

    table.add\_column("Right der. error", df\_right\_errors)

    table.add\_column("Central derivative", df\_central\_values)

    table.add\_column("Central der. error", df\_central\_errors)

    table.add\_column("Left second derivative", d2f\_left\_values)

    table.add\_column("Left second der. error", d2f\_left\_errors)

    # индексы элементов массива, в которых погрешность минимальна

    index\_r = numpy.argmin(df\_right\_errors)

    index\_c = numpy.argmin(df\_central\_errors)

    index\_2l = numpy.argmin(d2f\_left\_errors)

    print(

        "\nRight derivative(optimal result): ",

        df\_right\_values[index\_r],

        "\n\twith error ",

        df\_right\_errors[index\_r],

        "\n\tgot with step h =",

        h[index\_r],

    )

    print(

        "\nCentral derivative(optimal result): ",

        df\_central\_values[index\_c],

        "\n\twith error ",

        df\_central\_errors[index\_c],

        "\n\tgot with step h =",

        h[index\_c],

    )

    print(

        "\nLeft second derivative(optimal result): ",

        d2f\_left\_values[index\_2l],

        "\n\twith error ",

        d2f\_left\_errors[index\_2l],

        "\n\tgot with step h =",

        h[index\_2l],

    )

    # построение графиков погрешностей

    fig, ax = plt.subplots()

    plt.plot(

        numpy.log10(h),

        numpy.log10(df\_right\_errors),

        "o-b",

        label="Right derivative",

        lw=2,

        mec="b",

        mew=2,

        ms=5,

    )

    plt.plot(

        numpy.log10(h),

        numpy.log10(d2f\_left\_errors),

        "o-g",

        label="Left second derivative",

        lw=2,

        mec="g",

        mew=2,

        ms=5,

    )

    plt.plot(

        numpy.log10(h),

        numpy.log10(df\_central\_errors),

        "o-r",

        label="Central derivative",

        lw=2,

        mec="r",

        mew=2,

        ms=5,

    )

    plt.legend()

    ax.set\_xlabel('h')

    ax.set\_ylabel('error')

    plt.show()

    return table

a = 0

b = 10

c = 7

print("\nАналитическое значение f'(%s) =" % (c), df(c, 1))

print("\nАналитическое значение f''(%s) =" % (c), df(c, 2))

print(count\_arrays(c))

# %%

**Вывод формулы 8а:**

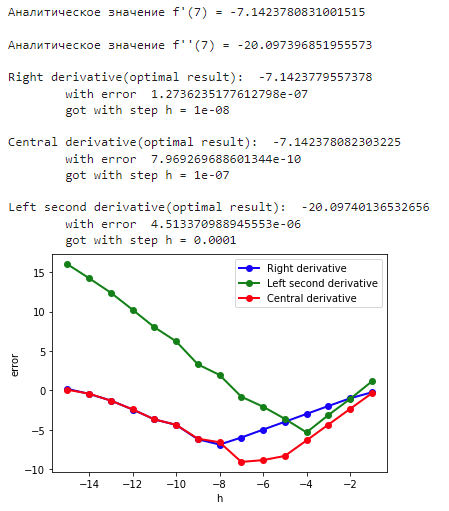
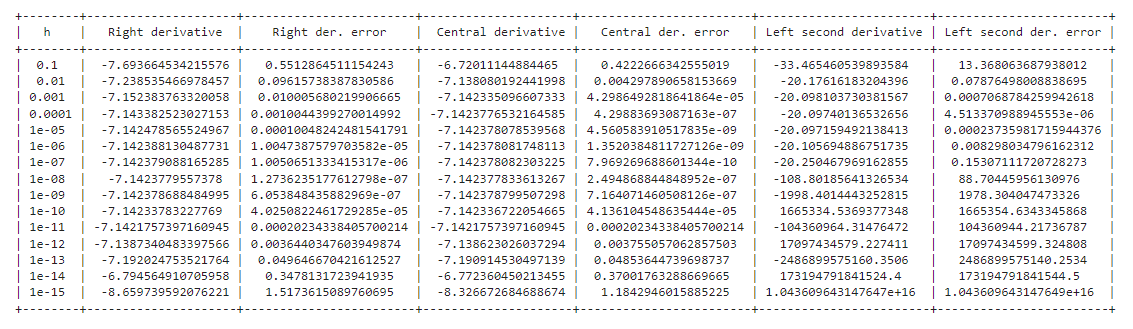
(**8**)  **(8a) остаточный** член 

Подставляем соответствующие разложения Тейлора:

Группируя по степеням h, получаем:

Следовательно,

**Результаты выполнения:**

****

*P.S. графики построены в шкале log10, иначе смысл их построения исчезает*

**Анализ результатов:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО Солонин Егор Витальевич Группа А-14-19 Номер варианта 41**    Значения производных в :  (1)  (3)  (8) | | | |
|  | Первый результат  при шаге | Наилучший результат | Последний результат  при шаге |
| Формула (1) | -7.6937  0.5513 | при шаге | -8.6597  1.5174 |
| Формула (3) | -6.7201  0.4223 | при шаге | -8.3267  1.1843 |
| Формула (11) | -33.4655  13.3681 | при шаге |  |

**Постановка задачи:**

**Задача 6.2.** Найти приближенное решение задачи Коши для обыкновенного дифференциального уравнения (ОДУ) 1 порядка с точностью .

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:**

1. Найти аналитическое решение задачи 24 из РЗ.

2. Составить программу вычисления решения методом Эйлера с заданной точностью, используя правило Рунге. Найти решение задачи с точностью , число точек N и шаг, при котором точность достигается. Построить график решения.

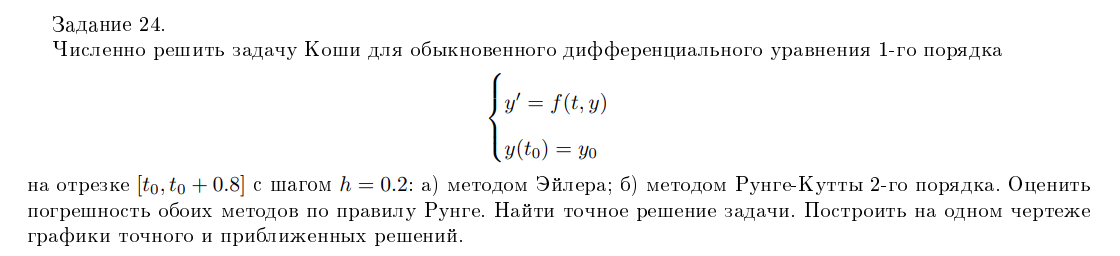
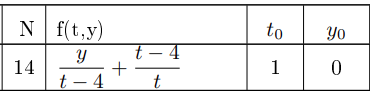
3. Составить программу вычисления решения с заданной точностью методом индивидуального варианта. Найти решение задачи с заданной точностью, число точек N

и шаг, при котором точность достигается. Построить график решения задачи.

4. Сравнить полученные результаты.

|  |  |
| --- | --- |
| 2, 15, 28,41,54 | Усовершенствованный Эйлера |

**Задача 24 (вариант 14):**

** **

**Теоретический материал:**

Метод Эйлера:

Усовершенствованный метод Эйлера:

**Решение задачи 24 (вариант 14):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | 1.2 | 1.4 | 1.6 | 1.8 |

**Метод Эйлера:**

**Оценка погрешности (метод Эйлера):**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 1.4 | 1.8 |

**Усовершенствованный метод Эйлера:**

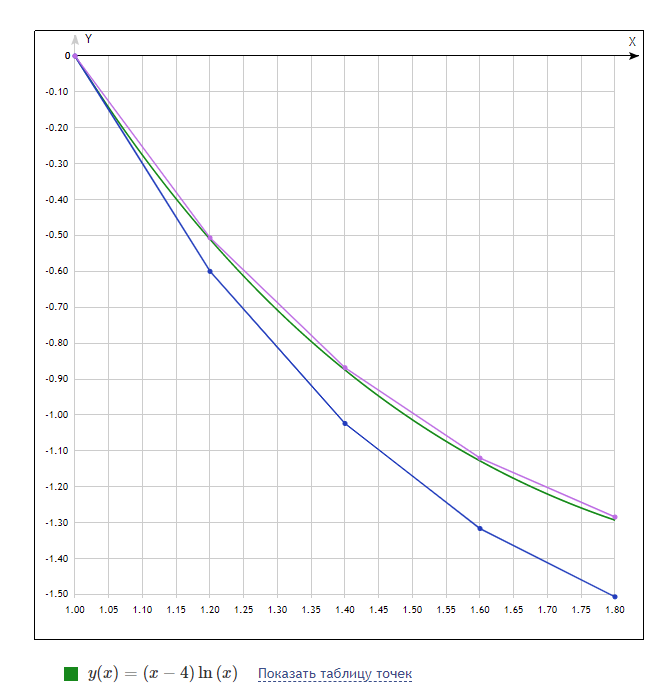
**Оценка погрешности (усовершенствованный метод Эйлера):**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 1.4 | 1.8 |

**Точное решение:**

**Итоговая таблица:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Точное решение |  |  |  |  |  |
| Метод Эйлера |  |  |  |  |  |
|  |  |  |  |  |  |
| Метод Эйлера |  |  |  |  |  |
| по Рунге |  |  |  |  |  |
| Усовершенствованный метод Эйлера |  |  |  |  |  |
|  |  |  |  |  |  |
| Усовершенствованный метод Эйлера |  |  |  |  |  |
| по Рунге |  |  |  |  |  |

**График:**

**\_\_ -** Метод Эйлера

\_\_ - Усовершенствованный метод Эйлера

**Код программы** *(пояснения к коду зеленым цветом)***:**

# %%

import numpy

import matplotlib.pyplot as plt

# задача Коши из задачи 24 типового расчета

dy = lambda y, t: y / (t - 4) + (t - 4) / t

# формула метода Эйлера

Euler = lambda y, t, h: y + h \* dy(y, t)

# формула усовершенствованного метода Эйлера

Euler\_modified = lambda y, t, h: y + h \* dy(y + h / 2 \* dy(y, t), t + h / 2)

# расчет числа точек N для шага h

get\_n = lambda h: int(numpy.ceil((t\_n - t\_0) / h))

# получить массив решений выбранным методом с шагом h

def get\_y\_data(h, Method):

    n = get\_n(h)

    t\_data = numpy.linspace(t\_0, t\_n, n + 1)

    y\_data = [0]

    for i in range(1, len(t\_data)):

        y\_data.append(Method(y\_data[i - 1], t\_data[i - 1], h))

    return y\_data

# проверка достижения заданной точности по праивлу Рунге

def Runge(y\_data\_h, y\_data\_2h):

    r\_data = [0]

    for i in range(1, len(y\_data\_2h)):

        r\_data.append(numpy.abs((y\_data\_h[2 \* i] - y\_data\_2h[i])))

    return False if max(r\_data) > epsilon else True

# функция, осуществляющая поиск решения выбранным методом,

# шага h, при котором заданная точность достигается;

# выполняющая построение графика решения выбранным методом

def solve(Method, methodName, plt):

    y\_data\_h = []

    h = 0.2

    while True:

        y\_data\_h = get\_y\_data(h, Method)

        y\_data\_2h = get\_y\_data(2 \* h, Method)

        if Runge(y\_data\_h, y\_data\_2h):

            break

        h /= 2

    n = get\_n(h)

    print("\n", methodName, ':')

    print("h = ", h)

    print("n = ", n)

    t\_data = numpy.linspace(t\_0, t\_n, n + 1)

    plt.plot(t\_data, y\_data\_h, label=methodName)

t\_0 = 1

t\_n = 1.8

epsilon = 10 \*\* (-6)

plt.subplots()

solve(Euler, "Euler", plt)

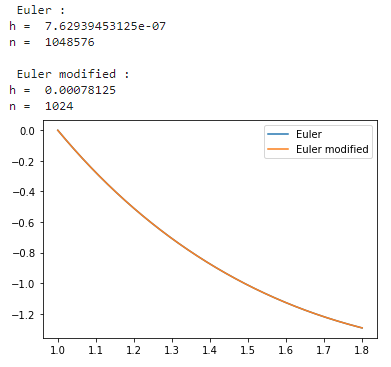
solve(Euler\_modified, "Euler modified", plt)

plt.legend()

plt.show()

# %%

**Результаты выполнения:**

****

**Анализ результатов:**

Метод Эйлера очевидно имеет меньшую точность. Для достижения заданной точности требуется выбрать шаг порядка . При этом получаем, что число разбиений находится в промежутке Более точно получить число разбиений нет возможности из-за ограничений вычислительных возможностей компьютера.

Усовершествованный метод Эйлера дает бОльшую точность. Точность достигается при шаге порядка . Число разбиений находится в промежутке

Алгоритм поиска оптимального шага:

1. проверка выполнения правила Рунге для каждой точки
2. В случае невыполнения правила Рунге хотя бы для одной точки – уменьшение шага в 2 раза