#### Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

#### Федеральное государственное бюджетное образовательное учреждение высшего образования

#### «Сибирский государственный университет телекоммуникаций и информатики» СибГУТИ

#### 09.03.01 "Информатика и вычислительная техника" профиль "Программное обеспечение средств вычислительной

#### техники и автоматизированных систем"

## ОТЧЁТ

### по курсовой работе по дисциплине

### «ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА»

### Выполнил:

### студент гр. ИП-116 /Хухарев Д.Г./

### «29» мая 2023 г.

### Преподаватель

### ассистент каф. ПМиК /Дементьева К.И./

### «29» мая 2023г. Оценка \_

### Новосибирск, 2023г.

1

**Содержание**

2

1. [Задание на курсовую работу 3](#_TOC_250004)
2. [Теория 4](#_TOC_250003)
3. [Листинг 9](#_TOC_250002)
4. [Результат работы 12](#_TOC_250001)
5. [Вывод 13](#_TOC_250000)

**Задание на курсовую работу**

Решить дифференциальное уравнение на интервале [0,1] методами Рунге кутта 2 и 4 порядка с точностью стартовый шаг h=0,1

y(0) = 1

y(1) = 2,718

Про интерполировать найденное решение с помощью интерполяционного многочлена ньютона по узлам интерполяции 0,0; 0,2; 0,4; 0,6; 0,8; 1,0.

**Теория**

**Метод Рунге-Кутта 2-го порядка**

Метод Рунге-Кутта второго порядка (RK2) является одним из наиболее распространенных численных методов для решения обыкновенных дифференциальных уравнений. Он основан на идее аппроксимации решения уравнения на каждом шаге с помощью касательной к кривой решения в данной точке.

Алгоритм метода RK2 состоит из следующих шагов:

1. Задаем начальное условие y(0) и шаг h.

2. Вычисляем первое приближение y1 = y(0) + h\*f(y(0)), где f(y) - правая часть дифференциального уравнения.

3. Вычисляем второе приближение y2 = y(0) + h\*f(y1).

4. Получаем окончательное значение решения на шаге h: y(h) = 1/2\*(y1 + y2) + O(h^3).

5. Повторяем шаги 2-4 для всех оставшихся шагов до конечной точки t.

Метод RK2 имеет второй порядок точности, что означает, что ошибка аппроксимации решения на каждом шаге пропорциональна h^2. Это делает метод RK2 достаточно точным для многих практических приложений, но для более сложных задач может потребоваться использование методов более высокого порядка.

**Метод Рунге-Кутта 4го порядка:**

Метод Рунге-Кутта четвертого порядка (RK4) является более точным и распространенным методом, чем RK2. Он также основан на аппроксимации решения уравнения на каждом шаге с помощью касательных, но использует более сложную формулу, которая позволяет достичь четвертого порядка точности.

Алгоритм метода RK4 состоит из следующих шагов:

1. Задаем начальное условие y(0) и шаг h.

2. Вычисляем первый приближенный шаг:

k1 = h\*f(y(0))

y1 = y(0) + 1/2\*k1

3. Вычисляем второй приближенный шаг:

k2 = h\*f(y1)

y2 = y(0) + 1/2\*k2

4. Вычисляем третий приближенный шаг:

k3 = h\*f(y2)

y3 = y(0) + k3

5. Вычисляем четвертый приближенный шаг:

k4 = h\*f(y3)

6. Получаем окончательное значение решения на шаге h:

y(h) = y(0) + 1/6\*(k1 + 2\*k2 + 2\*k3 + k4) + O(h^5)

7. Повторяем шаги 2-6 для всех оставшихся шагов до конечной точки t.

Метод RK4 имеет четвертый порядок точности, что означает, что ошибка аппроксимации решения на каждом шаге пропорциональна h^4. Это делает метод RK4 более точным, чем RK2, и позволяет использовать его для более сложных задач. Однако он также требует большего количества вычислений на каждом шаге, что может замедлить процесс решения.

**Метод кубических сплайнов** - это метод интерполяции функции, который используется для аппроксимации непрерывной функции на отрезке [a, b] кусочно-кубическими функциями. Кусочно-кубическая функция - это функция, которая на каждом отрезке [xi, xi+1] является многочленом третьей степени.

Для построения кубического сплайна необходимо решить систему линейных уравнений, составленную из условий непрерывности первой и второй производных на каждом узле сетки и условий равенства первой и последней производных на концах отрезка [a, b].

После решения системы линейных уравнений получаем коэффициенты кусочно-кубической функции на каждом отрезке [xi, xi+1]. Для вычисления значения функции в точке x необходимо определить на каком отрезке [xi, xi+1] она находится и вычислить значение кусочно-кубической функции на этом отрезке.

Метод кубических сплайнов имеет ряд преимуществ перед другими методами интерполяции, такими как высокая точность и гладкость полученной функции, возможность вычисления производных и интегралов, а также возможность использования для интерполяции функций с разрывами и особенностями.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | **class** **CubicSpline**:  **def** **\_\_init\_\_**(self, x\_values, y\_values):  self.xData = x\_values  self.yData = y\_values  n = len(self.xData)  self.h = [self.xData[i + **1**] - self.xData[i] **for** i **in** range(n - **1**)]  self.alpha = [**0.0**] \* (n - **1**)  self.l = [**0.0**] \* n  self.u = [**0.0**] \* (n - **1**)  self.z = [**0.0**] \* n  self.c = [**0.0**] \* n  self.b = [**0.0**] \* (n - **1**)  self.d = [**0.0**] \* (n - **1**)  **for** i **in** range(**1**, n - **1**):  self.alpha[i] = **3.0** \* ((self.yData[i + **1**] - self.yData[i]) / self.h[i] - (self.yData[i] - self.yData[i - **1**]) / self.h[i - **1**])  self.l[**0**] = **1.0**  self.u[**0**] = **0.0**  self.z[**0**] = **0.0**  **for** i **in** range(**1**, n - **1**):  self.l[i] = **2.0** \* (self.xData[i + **1**] - self.xData[i - **1**]) - self.h[i - **1**] \* self.u[i - **1**]  self.u[i] = self.h[i] / self.l[i]  self.z[i] = (self.alpha[i] - self.h[i - **1**] \* self.z[i - **1**]) / self.l[i]  self.l[n - **1**] = **1.0**  self.z[n - **1**] = **0.0**  self.c[n - **1**] = **0.0**  **for** j **in** range(n - **2**, -**1**, -**1**):  self.c[j] = self.z[j] - self.u[j] \* self.c[j + **1**]  self.b[j] = (self.yData[j + **1**] - self.yData[j]) / self.h[j] - self.h[j] \* (self.c[j + **1**] + **2.0** \* self.c[j]) / **3.0**  self.d[j] = (self.c[j + **1**] - self.c[j]) / (**3.0** \* self.h[j])    **def** **interpolate**(self, x):  n = len(self.xData)  index = **0**  **for** i **in** range(n - **1**):  **if** x >= self.xData[i] **and** x <= self.xData[i + **1**]:  index = i  **break**  delta\_x = x - self.xData[index]  interpolated\_value = (  self.yData[index]  + self.b[index] \* delta\_x  + self.c[index] \* delta\_x \*\* **2**  + self.d[index] \* delta\_x \*\* **3**  )  **return** interpolated\_value |

Этот код реализует кубическую сплайн интерполяцию. Кубический сплайн - это метод интерполяции, который используется для нахождения гладкой кривой, проходящей через заданные точки.

В конструкторе класса CubicSpline инициализируются массивы xData и yData, содержащие соответственно значения x и y точек, через которые нужно провести кривую. Затем вычисляются значения h, alpha, l, u, z, c, b и d, которые используются в функции интерполяции.

Функция interpolate(x) принимает значение x, для которого нужно найти соответствующее значение y на кривой. Сначала находится интервал, в котором находится заданное значение x. Затем вычисляется значение y на кривой с помощью формулы, использующей значения b, c и d, вычисленные в конструкторе.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | **def** **f**(x, y):  **return** (math.exp(x) + y + (y - y \* x)) / **3.0**  **def** **runge\_kutta2**(x0, y0, x, h):  n = int((x - x0) / h)  y = y0  **print**("Метод Рунге-Кутта 2-го порядка:")  **print**(f"y({x0}) = {y}")  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(x0, y)  k2 = h \* f(x0 + h, y + k1)  y += **0.5** \* (k1 + k2)  x0 += h  **print**("y({:.6f}) = {:.6f}".format(x0, y))  **def** **runge\_kutta4**(x0, y0, x, h):  n = int((x - x0) / h)  y = y0  **print**("Метод Рунге-Кутта 4-го порядка:")  **print**(f"y({x0}) = {y}")  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(x0, y)  k2 = h \* f(x0 + **0.5** \* h, y + **0.5** \* k1)  k3 = h \* f(x0 + **0.5** \* h, y + **0.5** \* k2)  k4 = h \* f(x0 + h, y + k3)  y += (**1.0** / **6.0**) \* (k1 + **2** \* k2 + **2** \* k3 + k4)  x0 += h  **print**("y({:.6f}) = {:.6f}".format(x0, y))  x0 = **0.0**  y0 = **1.0**  x = **1.0**  h = **0.1**  runge\_kutta2(x0, y0, x, h)  **print**()  runge\_kutta4(x0, y0, x, h)  **print**() |

Этот участок кода решает дифференциальное уравнение, заданное функцией f(x, y), на заданном интервале [x0, x] с помощью методов Рунге-Кутта 2-го и 4-го порядка. Для каждого метода вычисляются значения функции y на интервале с шагом h и выводятся на экран. Затем полученные значения функции интерполируются кубическим сплайном и результаты выводятся на экран.

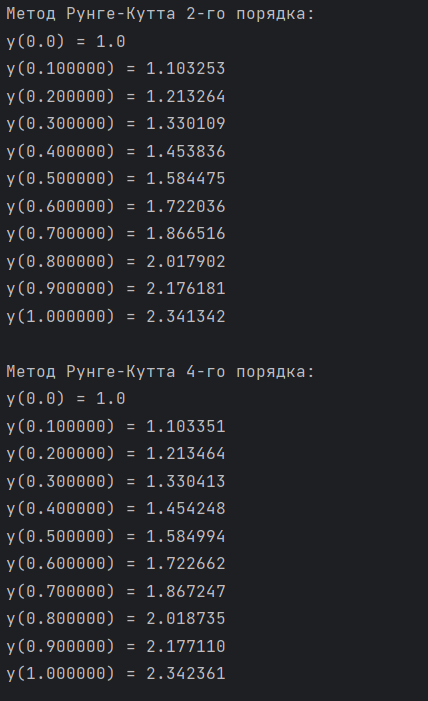
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | xData = [**0.0**, **0.2**, **0.4**, **0.6**, **0.8**, **1.0**]  yData = [y0]  xi = x0  yi = y0  n = int((x - x0) / h)  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(xi, yi)  k2 = h \* f(xi + **0.5** \* h, yi + **0.5** \* k1)  k3 = h \* f(xi + **0.5** \* h, yi + **0.5** \* k2)  k4 = h \* f(xi + h, yi + k3)  yi += (**1.0** / **6.0**) \* (k1 + **2** \* k2 + **2** \* k3 + k4)  xi += h  yData.append(yi)  **print**("Интерполяция кубическим сплайном:")  spline = CubicSpline(xData, yData)  **for** x **in** xData:  interpolatedValue = spline.interpolate(x)  **print**("P({:.5f}) = {:.5f}".format(x, interpolatedValue)) |

В этом участке кода происходит вычисление значений функции y с использованием метода Рунге-Кутты 4-го порядка на заданном интервале [x0, x] с шагом h. Для каждого значения x вычисляется соответствующее значение y, которое добавляется в список yData. Затем происходит интерполяция полученных значений функции кубическим сплайном и вывод результатов на экран.

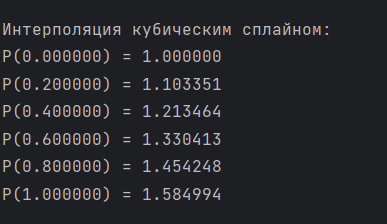
**Листинг**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132 | **import** **math**  **class** **CubicSpline**:  **def** **\_\_init\_\_**(self, x\_values, y\_values):  self.xData = x\_values  self.yData = y\_values  n = len(self.xData)  self.h = [self.xData[i + **1**] - self.xData[i] **for** i **in** range(n - **1**)]  self.alpha = [**0.0**] \* (n - **1**)  self.l = [**0.0**] \* n  self.u = [**0.0**] \* (n - **1**)  self.z = [**0.0**] \* n  self.c = [**0.0**] \* n  self.b = [**0.0**] \* (n - **1**)  self.d = [**0.0**] \* (n - **1**)  **for** i **in** range(**1**, n - **1**):  self.alpha[i] = **3.0** \* ((self.yData[i + **1**] - self.yData[i]) / self.h[i] - (self.yData[i] - self.yData[i - **1**]) / self.h[i - **1**])  self.l[**0**] = **1.0**  self.u[**0**] = **0.0**  self.z[**0**] = **0.0**  **for** i **in** range(**1**, n - **1**):  self.l[i] = **2.0** \* (self.xData[i + **1**] - self.xData[i - **1**]) - self.h[i - **1**] \* self.u[i - **1**]  self.u[i] = self.h[i] / self.l[i]  self.z[i] = (self.alpha[i] - self.h[i - **1**] \* self.z[i - **1**]) / self.l[i]  self.l[n - **1**] = **1.0**  self.z[n - **1**] = **0.0**  self.c[n - **1**] = **0.0**  **for** j **in** range(n - **2**, -**1**, -**1**):  self.c[j] = self.z[j] - self.u[j] \* self.c[j + **1**]  self.b[j] = (self.yData[j + **1**] - self.yData[j]) / self.h[j] - self.h[j] \* (self.c[j + **1**] + **2.0** \* self.c[j]) / **3.0**  self.d[j] = (self.c[j + **1**] - self.c[j]) / (**3.0** \* self.h[j])    **def** **interpolate**(self, x):  n = len(self.xData)  index = **0**    **for** i **in** range(n - **1**):  **if** x >= self.xData[i] **and** x <= self.xData[i + **1**]:  index = i  **break**  delta\_x = x - self.xData[index]  interpolated\_value = (  self.yData[index]  + self.b[index] \* delta\_x  + self.c[index] \* delta\_x \*\* **2**  + self.d[index] \* delta\_x \*\* **3**  )  **return** interpolated\_value  **def** **f**(x, y):  **return** (math.exp(x) + y + (y - y \* x)) / **3.0**  **def** **runge\_kutta2**(x0, y0, x, h):  n = int((x - x0) / h)  y = y0  **print**("Метод Рунге-Кутта 2-го порядка:")  **print**(f"y({x0}) = {y}")  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(x0, y)  k2 = h \* f(x0 + h, y + k1)  y += **0.5** \* (k1 + k2)  x0 += h  **print**("y({:.6f}) = {:.6f}".format(x0, y))  **def** **runge\_kutta4**(x0, y0, x, h):  n = int((x - x0) / h)  y = y0  **print**("Метод Рунге-Кутта 4-го порядка:")  **print**(f"y({x0}) = {y}")  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(x0, y)  k2 = h \* f(x0 + **0.5** \* h, y + **0.5** \* k1)  k3 = h \* f(x0 + **0.5** \* h, y + **0.5** \* k2)  k4 = h \* f(x0 + h, y + k3)  y += (**1.0** / **6.0**) \* (k1 + **2** \* k2 + **2** \* k3 + k4)  x0 += h  **print**("y({:.6f}) = {:.6f}".format(x0, y))  x0 = **0.0**  y0 = **1.0**  x = **1.0**  h = **0.1**  runge\_kutta2(x0, y0, x, h)  **print**()  runge\_kutta4(x0, y0, x, h)  **print**()  xData = [**0.0**, **0.2**, **0.4**, **0.6**, **0.8**, **1.0**]  yData = [y0] # Начальное значение y  xi = x0  yi = y0  n = int((x - x0) / h)  **for** i **in** range(**1**, n + **1**):  k1 = h \* f(xi, yi)  k2 = h \* f(xi + **0.5** \* h, yi + **0.5** \* k1)  k3 = h \* f(xi + **0.5** \* h, yi + **0.5** \* k2)  k4 = h \* f(xi + h, yi + k3)  yi += (**1.0** / **6.0**) \* (k1 + **2** \* k2 + **2** \* k3 + k4)  xi += h  yData.append(yi)  **print**("Интерполяция кубическим сплайном:")  spline = CubicSpline(xData, yData)  **for** x **in** xData:  interpolatedValue = spline.interpolate(x)  **print**("P({:.5f}) = {:.5f}".format(x, interpolatedValue)) |

**Результат работы**



(Рис. 1 Метод Рунге-Кутта 2-ого и 4-ого порядка)



(Рис. 2 Интерполяция кубическим сплайном)

**Вывод**

В ходе курсовой работы я улучшил свои навыки программирования на языке Python. Подробнее изучил такие математические алгоритмы как: метод Рунге-Кутта 2-ого и 4-ого порядка, а также метод интерполяции кубическими сплайнами.

Результатом работы стал рабочий код, который решает заданное дифференциальное уравнение с заданными характеристиками (начальное значение x, начальное значение y, конечное значение x, шаг h) и интерполирует его найденное решение с помощью кубического сплайна по узлам интерполяции 0,0, 0,2, 0,4, 0,6, 0,8, 1,0.