|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа № 6**

|  |  |
| --- | --- |
| **Тема Алгоритмы численного дифференцирования**  **Студент Якуба Д. В.**  **Группа ИУ7-43Б**  **Оценка (баллы)**  **Преподаватель Градов В. М.** |  |

Москва.

2020 г.

Оглавление

[Лабораторная работа по теме «Алгоритмы численного интегрирования» 3](#_Toc40013620)

[Тема: 3](#_Toc40013621)

[Цель работы: 3](#_Toc40013622)

[Задание: 3](#_Toc40013623)

[Входные данные: 3](#_Toc40013624)

[Выходные данные: 3](#_Toc40013625)

[Описание алгоритма 3](#_Toc40013626)

[Результат 5](#_Toc40013627)

[Алгоритм вычисления n корней полинома Лежандра n-ой степени 5](#_Toc40013628)

[Влияние количества выбираемых узлов сетки по каждому направлению на точность расчётов 6](#_Toc40013629)

[График зависимости 7](#_Toc40013630)

[Контрольные вопросы 7](#_Toc40013631)

[1. В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается? 7](#_Toc40013632)

[2. Построить формулу Гаусса численного интегрирования при одном узле. 7](#_Toc40013633)

[3. Построить формулу Гаусса численного интегрирования при двух узлах. 8](#_Toc40013634)

[4. Получить обобщенную кубатурную формулу, аналогичную (6.6) из лекции №6, для вычисления двойного интеграла методом последовательного интегрирования на основе формулы трапеций с тремя узлами по каждому направлению. 8](#_Toc40013635)

[Код программы 9](#_Toc40013636)

Лабораторная работа по теме «Алгоритмы численного дифференцирования»

Тема:

Построение и программная реализация алгоритмов численного дифференцирования.

Цель работы:

Получение навыков построения алгоритма вычисления производных от сеточных функций.

Задание:

Задана табличная функция. Имеется информация, что закономерность, представленная этой таблицей, может быть описана формулой:

Параметры функции неизвестны и определять их не нужно.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | y | 1 | 2 | 3 | 4 | 5 |
| 1 | 0.571 |  |  |  |  |  |
| 2 | 0.889 |  |  |  |  |  |
| 3 | 1.091 |  |  |  |  |  |
| 4 | 1.231 |  |  |  |  |  |
| 5 | 1.333 |  |  |  |  |  |
| 6 | 1.412 |  |  |  |  |  |

Вычислить первые разностные производные от функции и занести их в столбцы таблицы:

1. Односторонняя разностная производная

2. Центральная разностная производная.

3. Вторая формула Рунге с использованием односторонней производной

4. Введены выравнивающие переменные

5. Вторая разностная производная.

Входные данные:

Заданная таблица

Выходные данные:

Заполненная таблица с краткими комментариями по поводу использованных формул и их точности.

Описание алгоритма

Одним из наиболее универсальных методов построения формул численного дифференцирования заданных порядков точности относительно шага таблицы является метод разложения в ряды Тейлора.

Таблица задана на множестве значений аргумента, которые при постоянном шаге *h* образуют сетку , точки называют узлами сетки.

Если выполнить разложение функции в ряд Тейлора, приняв за центр разложения точку то получим разностные формулы для вычисления первых производных:

Или

Первое представленное выражение является правой разностной производной, а второе – левой разностной производной. В них мы имеем дело с самым низким порядком точности относительно шага.

Вычитая разложения можем прийти к центральной формуле для первой производной:

Такая формула более точная, а порядок точности уже второй.

Разностный аналог второй производной:

Погрешность вышеприведённых формул имеет вид:

Где - некоторая функция. Если некоторая приближённая формула Ф для вычисления величины имеет структуру:

То записав (6) для сетки с шагом *mh,* получим:

И по разложениям придём к первой формуле Рунге:

Комбинируя (2) и (3), получим вторую формулу Рунге, позволяющую за счёт расчёта на двух сетках с отличающимися шагами получить решение с более высокой точностью, чем заявленная теоретическая точность используемой формулы:

Формулы Рунге справедливы не только для операции дифференцирования, но и для любых других приближённых вычислений.

Ко всему вышеперечисленному следует также описать метод ввода выравнивающих переменных. При удачном подборе таких переменных исходная кривая может быть преобразована в прямую линию, производная от которой вычисляется точно по самым формулам. Пусть задана некоторая функция , и введены выравнивающие переменные и . Тогда, возврат к заданным переменным осуществляется этой формулой:

При этом можно вычислить по одной из односторонних формул.

Результат

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | y | 1 | 2 | 3 | 4 | 5 |
| 1 | 0.571 | - | - | 0.376 | 0.408 | - |
| 2 | 0.889 | 0.318 | 0.260 | 0.233 | 0.247 | -0.116 |
| 3 | 1.091 | 0.202 | 0.171 | 0.159 | 0.165 | -0.062 |
| 4 | 1.231 | 0.140 | 0.121 | 0.113 | 0.118 | -0.038 |
| 5 | 1.333 | 0.102 | 0.090 | - | 0.089 | -0.023 |
| 6 | 1.412 | 0.079 | - | - | - | - |

1. В вычислениях фигурировала левосторонняя формула, отчего и отсутствует значение при . Точность - *O(h).*

2. В вычислениях фигурирует центральная формула. Точность - .

3.

Контрольные вопросы

1.

2.

3.

Код программы

table = [[1, 2, 3, 4, 5, 6], [0.571, 0.889, 1.091, 1.231, 1.333, 1.412]]   
  
  
def leftSideDerivative(yValues):  
 answer = ['-', ]  
  
 for i in range(1, len(yValues)):  
 answer.append(yValues[i] - yValues[i - 1])  
  
 return answer  
  
  
def centerDerivative(yValues):  
 answer = ['-']  
  
 for i in range(1, len(yValues) - 1):  
 answer.append((yValues[i + 1] - yValues[i - 1]) / 2)  
  
 answer.append('-')  
 return answer  
  
  
def diffSecondDerivative(yValues):  
 answer = ['-']  
  
 for i in range(1, len(yValues) - 1):  
 answer.append((yValues[i - 1] - 2 \* yValues[i] + yValues[i + 1]))  
  
 answer.append('-')  
 return answer  
  
  
def rungeDerivativeLeft(yValues):  
 answer = ['-', '-']  
  
 for i in range(2, len(yValues)):  
 fStepY = yValues[i] - yValues[i - 1]  
 sStepY = (yValues[i] - yValues[i - 2]) / 2  
 answer.append(fStepY + fStepY - sStepY)  
  
 return answer  
  
  
def rungeDerivativeRight(yValues):  
 answer = []  
  
 for i in range(len(yValues) - 2):  
 fStepY = yValues[i + 1] - yValues[i]  
 sStepY = (yValues[i + 2] - yValues[i]) / 2  
 answer.append(fStepY + fStepY - sStepY)  
  
 answer.append("-")  
 answer.append("-")  
 return answer  
  
  
def varsDerivative(yValues, xValues):  
 result = []  
  
 for i in range(len(yValues) - 1):  
 curEtha = (1 / yValues[i] - 1 / yValues[i + 1]) / (1 / xValues[i] - 1 / xValues[i + 1])  
 result.append(yValues[i] \* yValues[i] \* curEtha / (xValues[i] \* xValues[i]))  
  
 result.append('-')  
 return result  
  
  
def main():  
 yValues = table[1]  
 xValues = table[0]  
  
 firstColumn = leftSideDerivative(yValues)  
 print("LeftSideDerivative:", firstColumn)  
  
 secondColumn = centerDerivative(yValues)  
 print("CenterDerivative:", secondColumn)  
  
 thirdColumnRight = rungeDerivativeRight(yValues)  
 print("RungeDerivativeRight:", thirdColumnRight)  
  
 thirdColumnLeft = rungeDerivativeLeft(yValues)  
 print("RungeDerivativeLeft:", thirdColumnLeft)  
  
 fourthColumn = varsDerivative(yValues, xValues)  
 print("VarsDevirative:", fourthColumn)  
  
 fifthColumn = diffSecondDerivative(yValues)  
 print("DiffSecondDerivative:", fifthColumn)  
  
  
main()