Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Кубанский государственный университет»

Факультет компьютерных технологий и прикладной математики

Кафедра прикладной математики

**ОТЧЕТ О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ**

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА   
(получение первичных навыков научно-исследовательской работы)**

период с 06.07.2022 г. по 19.07.2022 г.

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

*(Ф.И.О. студента)*

студента 260 группы 2 курса ФКТиПМ

Направление подготовки   
02.03.02 Фундаментальная информатика и информационные технологии

Руководитель учебной практики

доцент кафедры прикладной математики

факультета компьютерных технологий

и прикладной математики, к.ф.-м.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Письменский А.В.

Оценка по итогам защиты практики: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Краснодар 2022 г.

1. ***Условие задачи***

Написать программу для решения нелинейного уравнения

где подынтегральную функцию f(y), параметры a, b, допустимую погрешность решения **,** начальное число отрезков N и начальное приближение x0 задает пользователь (можно в коде программы).

Способ численного решения нелинейного уравнения

и связанного с ним решения определенного интеграла

Решить нелинейное уравнение используя метод хорд (секущих), при это интегрирование осуществить с помощью формулы прямоугольников 2-го порядка.

1. ***Математическая постановка задачи***

**Метод средних прямоугольников** – метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции полиномом Лагранжа нулевого порядка, построенным в одной точке на каждом частичном отрезке . В качестве этой точки при использовании метода средних прямоугольников выбирают середину частичного отрезка.

График подынтегральной функции, будет заключаться в приближённом вычислении площади под графиком – суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота – значением подынтегральной функции в этих узлах (Рис. 1).

Chart, line chart

Description automatically generated

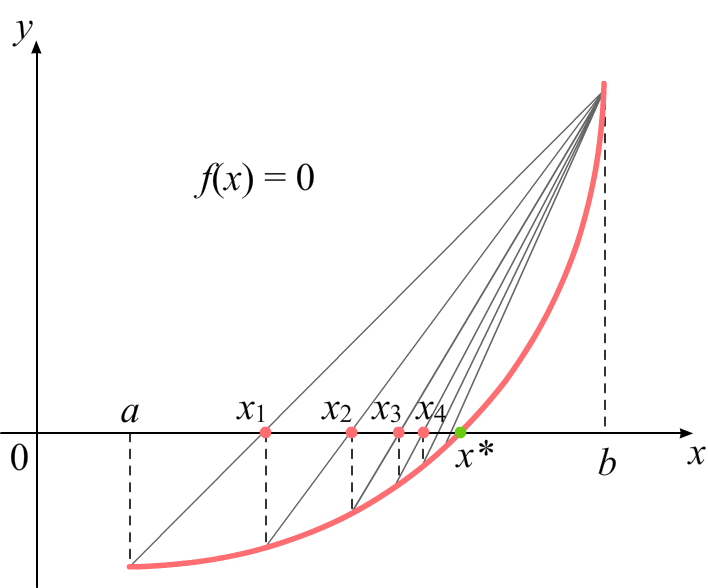
(Рис. 1)

Метод средних прямоугольников

Если отрезок является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле:

В случае разбиения отрезка интегрирования на элементарных отрезков приведённые выше формулы применяются на каждом из этих элементарных отрезков между двумя соседними узлами. В результате получаются составные квадратурные формулы:

**Метод хорд (секущих)** – алгоритм поиска корней, который использует последовательность корней секущих линий для лучшего приближения корня функции (Рис. 2).



(Рис. 2)

Метод хорд

Метод секущих определяется рекуррентным соотношением:

Как видно из рекуррентного соотношения, метод секущих требует двух начальных значений, x0 и x1, которые в идеале должны быть выбраны близко к корню.

**Ускорение параллельного алгоритма** – отношение времени выполнения последовательного алгоритма к времени выполнения параллельного алгоритма , где p – количество параллельных процессов:

**Загруженность параллельного алгоритма** – доля использования процессов, отношение ускорения параллельного алгоритма к количеству параллельных процессов :

1. ***Описание алгоритма решения***

В соответствии с условием поставленной задачи, требуется решить нелинейное уравнение (1), где необходимо найти . Для этого необходимо решить нелинейное уравнение (2), применяя способ численного решения нелинейного уравнения методом хорд (секущих). При этом вычисления определенного интеграла (3) будем производить, применяя формулу прямоугольников 2-го порядка.

Для выполнения поставленной задачи буду использовать язык программирования Python версии 3.8.5, а также Message Passing Interface (MPI) для распараллеливания программы.

1. ***Техническое описание программного продукта***

Функция отвечает за подынтегральную функцию – на вход принимает . Результат – значение, полученное путем подстановки в подынтегральную функцию.

Функции и являются реализацией метода численного интегрирования формулой прямоугольников 2-го порядка. Возвращают приближенное значение определенного интеграла на заданном отрезке.

Функция

реализует численное решение нелинейного уравнения методом хорд. Результатом будет значение при котором будет выполнено условия выхода соответствующий заданной точности .

Функция отвечает за установку начальных данных для уравнения (1) и его дальнейшего решения. Начальные данные, которые пользователь может изменить (в коде):

– нижний предел интегрирования (соответствует из формулы (1)).

– результат вычисления интеграла (соответствует из формулы (1)).

– количество разбиений отрезка . Требуется для вычисления интеграла.

– точность .

– начальное приближение.

Также, для распараллеливания программного продукта с помощью MPI, реализованы глобальные переменные, обозначающие количество доступных процессов, номер текущего процесса, завершение распараллеливания.

1. ***Листинг программы***

**import** **math**

**from** **mpi4py** **import** MPI

**import** **time**

**def** **f**(x):

**return** math.sin(x)\*math.cos(x)-math.sin(x)\*math.sin(x)+math.cos(x)

**def** **rectangle\_rule**(func, a, b, \_amount\_of\_iterations, multiplier):

step = (b - a) / \_amount\_of\_iterations

result\_sum = **0.0**

x\_start = a + multiplier \* step

**for** i **in** range(**1**, int(\_amount\_of\_iterations)):

result\_sum += func(x\_start + i \* step)

**return** result\_sum \* step

**def** **midpoint\_rectangle\_rule**(func, start, end, \_amount\_of\_iterations):

**return** rectangle\_rule(func, start, end, \_amount\_of\_iterations, **0.5**)

**def** **secant\_method**(start\_segment, end\_segment, epsilon, \_integral\_from, \_integral\_result, \_amount\_of\_iterations):

**while** abs(end\_segment-start\_segment) >= epsilon:

local\_amount\_of\_iterations = \_amount\_of\_iterations / p

step\_fx1 = (end\_segment - \_integral\_from) / \_amount\_of\_iterations

local\_start\_fx1 = \_integral\_from + my\_rank \* local\_amount\_of\_iterations \* step\_fx1

local\_end\_fx1 = local\_start\_fx1 + local\_amount\_of\_iterations \* step\_fx1

step\_fx2 = (start\_segment - \_integral\_from) / \_amount\_of\_iterations

local\_start\_fx2 = \_integral\_from + my\_rank \* local\_amount\_of\_iterations \* step\_fx2

local\_end\_fx2 = local\_start\_fx2 + local\_amount\_of\_iterations \* step\_fx2

integral\_fx1 = midpoint\_rectangle\_rule(f, local\_start\_fx1, local\_end\_fx1, local\_amount\_of\_iterations)

integral\_fx2 = midpoint\_rectangle\_rule(f, local\_start\_fx2, local\_end\_fx2, local\_amount\_of\_iterations)

**if** my\_rank == **0**:

fx1 = integral\_fx1

fx2 = integral\_fx2

**for** source **in** range(**1**, p):

integral\_array = comm.recv(source=source)

fx1 = fx1 + integral\_array[**0**]

fx2 = fx2 + integral\_array[**1**]

fx1 -= \_integral\_result

fx2 -= \_integral\_result

x\_temp = end\_segment

end\_segment = end\_segment - (end\_segment - start\_segment) \* fx1 / (fx1 - fx2)

start\_segment = x\_temp

**for** rank\_index **in** range(**1**, p):

comm.send([start\_segment, end\_segment], dest=rank\_index)

**else**:

comm.send([integral\_fx1, integral\_fx2], dest=**0**)

segment\_array = comm.recv(source=**0**)

start\_segment = segment\_array[**0**]

end\_segment = segment\_array[**1**]

**return** end\_segment

**def** **main**():

start\_time = time.perf\_counter()

integral\_from = **0**

integral\_result = **3**

amount\_of\_iterations = **100000**

epsilon = **0.00001**

x0 = -**4**

x1 = x0 + **0.5**

result = secant\_method(x0, x1, epsilon, integral\_from, integral\_result, amount\_of\_iterations)

end\_time = time.perf\_counter()

**if** my\_rank == **0**:

**print**(f"Result: {result}")

**print**(f"Time: {end\_time - start\_time}")

comm = MPI.COMM\_WORLD

my\_rank = comm.Get\_rank()

p = comm.Get\_size()

main()

MPI.Finalize

1. ***Тестирование программного продукта***

Для проверки правильности выполнения задачи, то есть решения уравнения (1), будем находить абсолютную погрешность между результатом работы программного продукта и верным ответом. В качестве верного ответа будем использовать результат решения уравнения вопросно-ответной системой WolframAlpha. При большом количестве цифр после запятой, значения будут обрезаны.

Пример 1

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | integral\_from | integral\_result | amount\_of\_iterations | epsilon | x0 |
| x\*x\*x-18\*x-83 | 8 | 10 | 240000 | 1e-5 | 2 |

Результаты работы программы:

1. Количество процессов: 1.

Text

Description automatically generated

1. Количество процессов: 5.

Text

Description automatically generated

1. Количество процессов: 24.

Text

Description automatically generated

Верный ответ:

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 5 | 24 |
| Погрешность | 6.319e-5 | 4.599e-5 | 4.578e-5 |

Пример 2

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | integral\_from | integral\_result | amount\_of\_iterations | epsilon | x0 |
| x\*x\*x+cos(x) | 5 | 7 | 240000 | 1e-5 | 5 |

Результаты работы программы:

1. Количество процессов: 1.

Text

Description automatically generated

1. Количество процессов: 5.



1. Количество процессов: 24.

Text

Description automatically generated

Результат вычислений WolframAlpha:

Верный ответ:

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 5 | 24 |
| Погрешность | 2.21e-7 | 1.121e-6 | 5.401e-6 |

Пример 3

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | integral\_from | integral\_result | amount\_of\_iterations | epsilon | x0 |
| sin(x)\*cos(x)-sin(x)\*sin(x)+ cos(x) | 0 | 3 | 100000 | 1e-5 | -4 |

Результаты работы программы:

1. Количество процессов: 1.



1. Количество процессов: 5.



1. Количество процессов: 24.



Верный ответ:

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 5 | 24 |
| Погрешность | 2.408e-5 | 5.291e-5 | 6.903e-4 |

1. ***Эффективность распараллеливания***

Для тестирования эффективности распараллеливания будем

использовать среднее арифметическое времени 5-ти испытаний для каждого варианта установки количества процессов. Учитывая большое количество возможных вариантов установки количества процессов, будем заносить в таблицу данные до тех пор, пока время выполнения программы не начнет увеличиваться.

Для наглядности время будем округлять до 2-х цифр после целой части.

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | integral\_from | integral\_result | amount\_of\_iterations | epsilon | x0 |
| x\*x\*x-18\*x-83 | 8 | 10 | 1000000 | 1e-5 | 2 |

Таблица зависимости времени выполнения программы от числа процессов:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Время, сек. | 2.67 | 1.38 | 0.93 | 0.69 | 0.56 | 0.48 | 0.42 | 0.37 | 0.33 |
| Кол-во процессов | 10 | 11 | 12 | 13 | 14 |
| Время, сек. | 0.3 | 0.28 | 0.28 | 0.32 | 0.31 |

Таблица зависимости ускорения работы программы от числа процессов:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Ускорение | 1 | 1.93 | 2.87 | 3.86 | 4.76 | 5.56 | 6.35 | 7.21 | 8.09 |
| Кол-во процессов | 10 | 11 | 12 | 13 | 14 |
| Ускорение | 8.9 | 9.53 | 9.53 | 8.34 | 8.61 |

Таблица загруженности:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Загруженность | 1 | 0.96 | 0.95 | 0.96 | 0.95 | 0.92 | 0.9 | 0.9 | 0.89 |
| Кол-во процессов | 10 | 11 | 12 | 13 | 14 |
| Загруженность | 0.89 | 0.86 | 0.79 | 0.64 | 0.61 |

1. **Использованные источники**

https://www.wolframalpha.com

https://en.wikipedia.org/wiki/Secant\_method

https://ru.wikipedia.org/wiki/Метод\_Ньютона

https://habr.com/en/post/121925/

http://statistica.ru/branches-maths/chislennye-metody-resheniya-uravneniy/