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

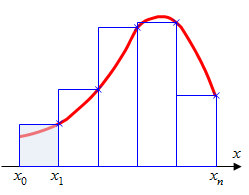
где подынтегральную функцию , параметры , , допустимую погрешность решения **,** начальное число отрезков и начальное приближение задает пользователь (можно в коде программы).

Способ численного решения нелинейного уравнения

и связанного с ним решения определенного интеграла

Решить нелинейное уравнение используя канонический метод Ньютона, при этом интегрирование осуществить с помощью формулы прямоугольников 2-го порядка(метод правых прямоугольники).

1. ***Математическая постановка задачи***

**Метод прямоугольников** – метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод 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)

Так значение интеграла на заданном отрезке [a, b] можно найти по формуле (Рис.1) *правых прямоугольников*:

В случае разбиения отрезка интегрирования на элементарных отрезков приведённая выше формула применяется на каждом из этих элементарных отрезков между двумя соседними узлами. В результате получается составная квадратурная формула:

**Метод Ньютона** (метод касательных) – это итерационный численный метод нахождения корня (нуля) заданной функции.

Изображение выглядит как лазер, темный

Автоматически созданное описаниеОсновная идея метода заключается в следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к графику исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность.

(Рис. 2)

Рассмотрим Рис.2. Синим изображена функция ,ноль которой необходимо найти, красным – касательная в точке очередного приближения . Здесь мы можем увидеть, что последующее приближение лучше предыдущего .

*Алгоритм:*

Задается начальное приближение . Пока не выполнено условие остановки, в качестве которого можно взять или , вычисляют новое приближение: .

**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 x. Для этого необходимо решить нелинейное уравнение (2), применяя способ численного решения нелинейного уравнения методом Ньютона. При этом вычисления определенного интеграла (3) будем производить, применяя формулу прямоугольников 2-го порядка. Для выполнения поставленной задачи использован язык программирования Python версии 3.8.5, а также модуль mpi4py для реализации средств MPI. Этот модуль основан на спецификации MPI и предоставляет объектно-ориентированный интерфейс, напоминающий привязки C++ MPI-2. Он поддерживает двухточечную (отправляет, получает) и коллективную (широковещательную передачу, разброс, сбор) связь любого объекта Python, а также эффективную связь объектов Python, открывающих интерфейс буфера Python (например, массивы NumPy и встроенные байты/массивы).

Решать поставленную задачу будем следующем образом: В алгоритме на управляющем потоке будем методом Ньютона итерационно вычислять нелинейное уравнение (2). Остальным оставшимся потокам (хотя бы одному) будет дан определенный интеграл (3), который мы будем вычислять методом правых прямоугольников, где в свою очередь распределим интегрируемую функцию на отрезки и отдадим их данным нам потокам. Посчитанный определенный интеграл будет передан управляющему потоку, где будет найден результат нелинейного уравнения (1).

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

В программе описано 4 функции:

,

, ,

.

Функция df вычисляет значение математической функции для данного аргумента .

Функция вычисляет определенных интеграл для данной ей функции, в соответствии с заданным промежутком , и количеством итераций .

Функция newton реализует в себе параллельное вычисление интеграла с дальнейшим использованием результата его вычисления в нелинейной функции для подсчета ответа.

Функция main запрашивает значения у пользователя и выдает результат решаемой задачи. Подынтегральная функция указывается в коде явно. Значения вводятся пользователем в консоли по порядку: 1) предел ,

2) результат определенного интеграла , 3) количество итераций подсчета интеграла, 4) желаемая точность в методе, 5) начальное приближение .

1. ***Листинг программы***

from mpi4py import MPI  
import time  
import sys  
start\_time = time.perf\_counter()  
def df(x):  
 return x \* x \* x \* x - x \* x \* x - 18  
def rectangle\_rule\_right(func, a, b, num\_of\_iter):  
 step = (b - a) / num\_of\_iter  
 result\_sum = 0.0  
 x\_start = a+1.0\*step  
 for i in range(1, int(num\_of\_iter)):  
 result\_sum += func(x\_start + i \* step)  
 return result\_sum \* step  
def newton(a, b, x0, comm, eps=1e-7, kmax=10, total\_num\_iter= 2400000):  
 my\_rank = comm.Get\_rank()  
 if my\_rank == 0:  
 p = comm.Get\_size()  
 x = x0  
 x\_prev, i = x0 + 2 \* eps, 0  
 while abs(x - x\_prev) >= eps and i < kmax:  
 ab = (x - a) / (p - 1)  
 for j in range(1, p):  
 d = {'a': a + (j - 1) \* ab,  
 'b': a + j \* ab,  
 'num\_of\_iter': total\_num\_iter / (p - 1),  
 'is\_stop': False}  
 comm.send(d, dest=j)  
 integr\_x = 0  
 for j in range(1, p):  
 integr\_x += comm.recv(source=j)  
 x\_prev = x  
 x = x - (integr\_x - b) / df(x)  
 print(x, ' : ', x\_prev)  
 i += 1  
 for j in range(1, p):  
 d = {'is\_stop': True}  
 comm.send(d, dest=j)  
 return x  
 else:  
 while True:  
 d = comm.recv(source=0)  
 if d['is\_stop']:  
 break  
 else:  
 res = rectangle\_rule\_right(df, d['a'], d['b'], d['num\_of\_iter'])  
 comm.send(res, dest=0)  
def main():  
 comm = MPI.COMM\_WORLD  
 a, b, total\_num\_iter, epsilon, x0 = \  
 float(sys.argv[1]), float(sys.argv[2]), int(sys.argv[3]), float(sys.argv[4]), float(sys.argv[5])  
 res = newton(a=a, b=b, x0=0, eps=epsilon, kmax=100, comm=comm, total\_num\_iter=total\_num\_iter)  
 if comm.Get\_rank() == 0:  
 print('Result ! =>', res)  
 end\_time = time.perf\_counter()  
 print('Time ! =>', end\_time - start\_time)

MPI.Finalize  
main()

1. ***Тестирование программного продукта***

Для проверки правильности выполнения задачи, то есть решения уравнения , будем находить абсолютную погрешность между результатом работы программного продукта и верным ответом. В качестве эталонного верного ответа будем использовать результат решения уравнения в приложении Photomath.

Пример 1

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | a | b | iterations | epsilon | x0 |
|  | 8 | 2684 | 240000 | 1e-5 | 0 |

Правильный ответ, данный приложением Photomath: X=8,637992388…

Результат программы на 4 процессах:



Изображение выглядит как текст

Автоматически созданное описание

Результат программы на 3 процессах:

Изображение выглядит как текст

Автоматически созданное описание

Результат программы на 2 процессах:

Изображение выглядит как текст, знак, черный, закрыть

Автоматически созданное описание

Как видно, результат во всех случаях совпал с эталонным вплоть до заданной точности равной: 1e-5 (0.00001).

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 2 | 3 | 4 |
| Погрешность | 1.5696e-06 | 3.8435e-06 | 6.1221e-06 |

Пример 2

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | a | b | iterations | epsilon | x0 |
|  | 80 | 75 | 1440000 | 1e-7 | 0 |

Правильный ответ, данный приложением Photomath: X=42,49885

Результат программы на 4 процессах:



Изображение выглядит как текст, знак, черный

Автоматически созданное описание

Результат программы на 3 процессах:



Изображение выглядит как текст

Автоматически созданное описание

Результат программы на 2 процессах:



Изображение выглядит как текст

Автоматически созданное описание

В этот раз можно наглядно заметить корреляцию количества запускаемых процессов для поиска решения и точности этого решения (меньше процессов выше точность).

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 2 | 3 | 4 |
| Погрешность | 2.9467e-05 | 5.6856e-05 | 8.4244e-05 |

Пример 3

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | a | b | iterations | epsilon | x0 |
|  | 5 | 30 | 2280000 | 1e-8 | 0 |

Правильный ответ, данный приложением Photomath: X=7,402629

Результат программы на 4 процессах:



Изображение выглядит как текст, знак

Автоматически созданное описание

Результат программы на 3 процессах:



Изображение выглядит как текст, знак, часы, датчик

Автоматически созданное описание

Результат программы на 2 процессах:



Изображение выглядит как текст

Автоматически созданное описание

Таблица абсолютной погрешности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 2 | 3 | 4 |
| Погрешность | 9.9890e-07 | 2.0335e-06 | 3.0680e-06 |

1. ***Эффективность распараллеливания***

Для тестирования эффективности распараллеливания будем

использовать среднее арифметическое времени 5-ти испытаний для каждого варианта установки количества процессов. Тесты будем проводить на 4-х физических ядрах компьютера, где одно ядро всегда занято управляющей логикой и не используется для расчета интеграла.

Для наглядности время будем округлять до 3-х цифр после целой части.

Начальные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| f(x) | a | b | iterations | epsilon | x0 |
|  | 5 | 5 | 1600000 | 1e-8 | 0 |

Таблица зависимости времени выполнения программы от числа процессов:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 |
| Время, сек. | 4.000 | 2.01 | 1.422 |

Таблица зависимости ускорения работы программы от числа процессов:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 |
| Ускорение | 1 | 1.99 | 2.81 |

Таблица загруженности:

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | 1 | 2 | 3 |
| Загруженность | 1 | 0.995 | 0.936 |

**Использованные источники**

<https://ru.wikipedia.org/wiki/Метод_прямоугольников>

<https://ru.wikipedia.org/wiki/Метод_Ньютона>

https://www.sharcnet.ca/help/images/4/4b/Python\_mpi\_gis.pdf