Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина «Основы алгоритмизации и программирования»

К защите допустить:

Руководитель курсового проекта

ассистент кафедры ИТАС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Езовит

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

**«РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДАМИ ВЕГСТЕЙНА, ПАРАБОЛ И ДЕЛЕНИЯ ОТРЕЗКА ПОПОЛАМ»**

БГУИР КП 6-05-0612-03 006 ПЗ

Выполнил студент группы 328501

СУВОРОВ Владислав Валерьевич

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

(подпись студента)

Курсовой проект представлен на

проверку \_\_\_.\_\_\_\_.2024

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

(подпись студента)

Минск 2024

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ

Заведующий кафедрой

*­­*

2024г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    *Суворову Владиславу Валерьевичу*

1. Тема проекта Решение нелинейных уравнений методами Вегстейна, парабол и деления отрезка пополам

2. Срок сдачи студентом законченного проекта *8 июня* *2024г.–*

3. Исходные данные к проекту*:*

*1. Нелинейные уравнения имеют вид:*

*1)*

*2)*

*3)*

*2. Интервалы для уравнений*

*1) [4; 7]*

*2) [4; 8]*

*3) [2; 6]*

*3. Методы исследования:*

*Метод Вегстейна, метод параболы, метод деления отрезка пополам*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение.*

*1.Теория.*

*2.Программная реализация методов.*

*3.Анализ алгоритмов.*

*Заключение.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1.Схема алгоритма метода Вегстейна.*

*2. Схема алгоритма метода параболы.*

*3. Схема алгоритма метода деления отрезка пополам.*

6. Дата выдачи задания *19 февраля 2024г.*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*Раздел 1 к 15.03.2024– 30 %;–*

*раздел 2 к 15.04.2024– 25 %;–*

*раздел 3 к 15.05.2024 – 30 %;*

*оформление к – 15%.*

*Защита курсового проекта с по 2024г.–2*

РУКОВОДИТЕЛЬ

А.В. Езовит*.*

(подпись)

Задание принял к исполнению *\_\_\_\_\_\_\_\_ \_\_ В. В. Суворов*

(дата и подпись студента)

СОДЕРЖАНИЕ

[Введение 5](#_Toc168846174)

[1 Теория 6](#_Toc168846175)

[1.1 Метод простой итерации 6](#_Toc168846176)

[1.2 Метод Ньютона 7](#_Toc168846177)

[1.3 Метод секущих 8](#_Toc168846178)

[1.4 Метод Вегстейна 9](#_Toc168846179)

[1.5 Метод парабол 10](#_Toc168846180)

[1.6 Метод деления отрезка пополам 12](#_Toc168846181)

[2 Программная реализация методов 13](#_Toc168846182)

[2.1 Метод Вегстейна 16](#_Toc168846183)

[2.2 Метод деления отрезка пополам 16](#_Toc168846184)

[2.3 Метод параболы 17](#_Toc168846185)

[3 Анализ алгоритмов 19](#_Toc168846186)

[3.1 Скорость алгоритмов 19](#_Toc168846187)

[3.2 Эффективность алгоритмов 20](#_Toc168846188)

[3.3 Потребление ресурсов 21](#_Toc168846189)

[3.4 Общее сравнение 22](#_Toc168846190)

[3.5 Выводы 22](#_Toc168846191)

[Заключение 24](#_Toc168846192)

[Список использованной литературы 25](#_Toc168846193)

[Приложени А(обязательное) Код программы 26](#_Toc168846194)

ВВЕДЕНИЕ

Математические модели, основанные на функциональных зависимостях типа *y = f(x)*, играют ключевую роль в анализе и прогнозировании поведения физических процессов. Так понимание и нахождение корней уравнений необходимо для определения критических точек, критических значений параметров, а также для решения множества практических задач, начиная от точного моделирования поведения материалов до оптимизации конструкций в инженерных системах. Особую роль играют нелинейные уравнения, они являются одним из важных классов математических задач, требующих особого подхода к их решению. Эффективные методы решения нелинейных уравнений имеют широкое применение в различных областях науки и техники, начиная от физики и инженерии и заканчивая экономикой и биологией. Одной из таких задач является нахождения всех возможных значений *x*, при которых функция *f(x)* обращается в ноль, т.е. решение уравнения

где *f*(*x*) – нелинейная функция вида [1]:

– нелинейная алгебраическая функция (полином или многочлен)

– трансцендентная функция – тригонометрическая, обратная тригонометрическая, логарифмическая, показательная и гиперболическая функция;

– комбинирование этих функций, например ()

Точные значения *x* в таких уравнениях зачастую невозможно найти. Как раз для этого существуют так называемые численные методы, суть которых заключается в нахождении максимально приближенных значений.

Существует множество способов для данных вычислений. Многие поддаются ручному счету, однако с появление компьютерных технологий и программного обеспечения, способного эффективно решать сложные математические задачи, нахождение корней уравнения (1.1) стало более простой задачей.

В этой работе рассмотрены методы Вегстейна, парабол и деления отрезка пополам, с разбором их математической, алгоритмической и программной части.

1 ТЕОРИЯ

Нахождение корней уравнения (1.1) проходит в 2 этапа:  
1 Нахождение промежутков, в которых находятся корни уравнения.  
2 Вычисление приближенных значений с точность ε.

Первый этап заключается в вычислении таблицы значений на некотором отрезка [*a, b*] с некоторым шагом h. Далее по этим точкам строится график и определяются интервалы, в которых находятся корни. Если же построение графика является сложной или затратной задачей, то, исходя из пар аргумента и значения функции можно установить, где функция пересекает ось абсцисс. Если же на промежутке не наблюдается пересечений, то берется либо другой промежуток, в зависимости от характера функции, либо уменьшается шаг, для большего количества значений функции.

На втором этапе более точное значение в пределах εвычисляется одним из итерационных методов. На определенном интервале (α*,* β*)* где и находятся интересующие нас корни, выбирается *m* начальных значений , (обычно ) после чего находятся последовательные члены () рекуррентной последовательности порядка *m* по правилу до тех пора, пока не будет удовлетворено . Где в качестве искомого значение *x* выбирается приближение .

1.1 Метод простой итерации

Часто существуют представления уравнения (1.1) [2], выраженные относительно *x* по формуле

Стоит заметить, что переход от уравнения (1.1) к уравнению (2.1) можно осуществить путем некой зависимости от *x*, например как на формуле

Где – произвольная, непрерывная, знакопостоянная функция.

В таком случае корни уравнения (2.1) будут равны корням уравнения (1.1) и наоборот (рисунок 1.1).

Таким образом рекуррентные челны последовательности будут вычислять по закону

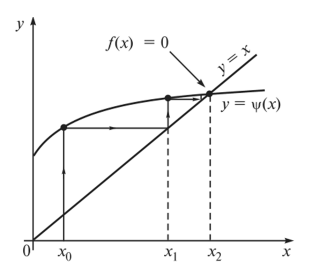


Рисунок 1.1 – Иллюстрация метода простой итерации.

1.2 Метод Ньютона

Метод Ньютона (рисунок 1.2) является модификацией метода простой итерации и имеет второе название метод касательных, из-за движения по осям *x* и *y* не по лестнице, а по построенным к точкам касательным. Его суть заключается в том, что если функция *f*(*x*) имтеет непрерывную производную, товыбрав в (2.2) значение , получим эквивалентное уравнение

Откуда форма рекуррентной сходимости метода Ньютона

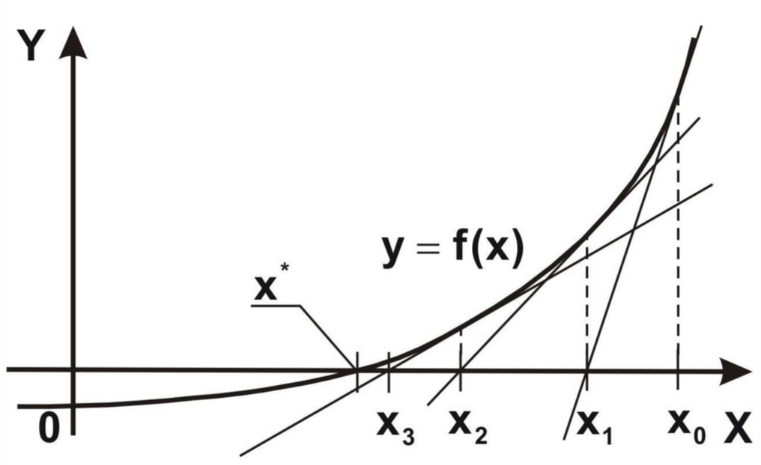


Рисунок 1.2 – Иллюстрация метода Ньютона.

1.3 Метод секущих

Данный метод (рисунок 1.3) является модификацией метода Ньютона, в котором вместо касательной проводится секущая, путем замены производной в (2.4) на ее приближенное значение, откуда

Где *h* – некоторый малый параметр.

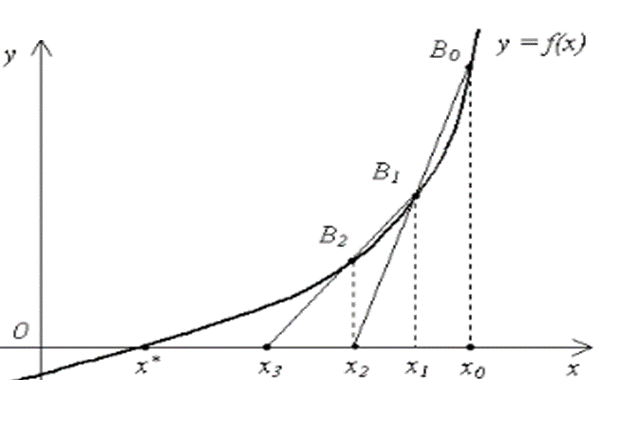


Рисунок 1.3 – Иллюстрация метода секущих.

1.4 Метод Вегстейна

Этот метод является модификацией метода секущих (рисунок 1.4). В нем предлагается вместо приближенной точки в (2.5) использовать , полученную в предыдущей итерации. Таким образом расчетная формула имеет вид

Для начала работы следует задать 2 значения приближения . Метод Вегстейна сходится медленнее, однако требует в 2 раза меньше вычислений *f(x)*, что дает ему большую эффективность.

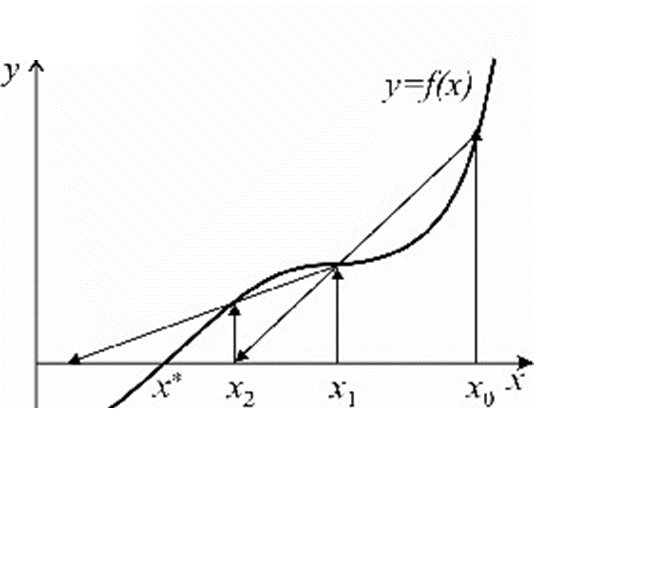


Рисунок 1.4 – Иллюстрация метода Вегстейна.

1.5 Метод парабол

Этот метод основан на замене исходной функции (1.1) интерполяционным многочленом второй степени (параболой), которая строится по 3 точкам, выбранным последовательно () (рисунок 1.5). В этом случае в качестве приближенного значения корня уравнения (1.1) используется точка пересечения параболы и оси абсцисс (*x*).

В общем виде многочлен записывается в таком виде

где *k* – точка, в которой функция заменяется на интерполяционный многочлен.

– коэффициенты уравнения, вычисляемые по следующим формулам:

Если рассматривать интервал [*a*, *b*], то параболу можно строить по 3 точкам: .

В соответствии с вышеописанным условие, многочлен будет иметь вид

Соответствующий ему квадратный вид

Коэффициенты уравнения вычисляются следующим образом

Таким образом интерполяционный многочлен второй степени пересекает ось абсцисс в двух точка, которые являются решение квадратного уравнения

После некоторых преобразований расчетной формулой двух точек пересечения параболы с осью абсцисс

Дальше из двух значений необходимо выбрать одно, которое будет находится в выбранном интервале [*a*, *b*]. Это значение и будет являться приближенным значением корня функции (1.1).



Рисунок 1.5 – Иллюстрация метода парабол.

1.6 Метод деления отрезка пополам

Если функция не является непрерывной и дифференцируемой вблизи искомого корня, для нахождения корней используется более медленный, но надежный метод деления отрезка пополам [3]. Его алгоритм очень прост и заключается в построении рекуррентной последовательности так, что в виде границ используется интервал (α, β), в котором точно находится искомый корень. Так начальные значения берутся , далее находится середина как новое значение . Значение будет находится исходя из новых границ [] или [], в зависимости от того, на каком именно интервале будет находится корень (рисунок 1.6). Последующие значения будут находится по такому же принципу до достижения требуемой точности .

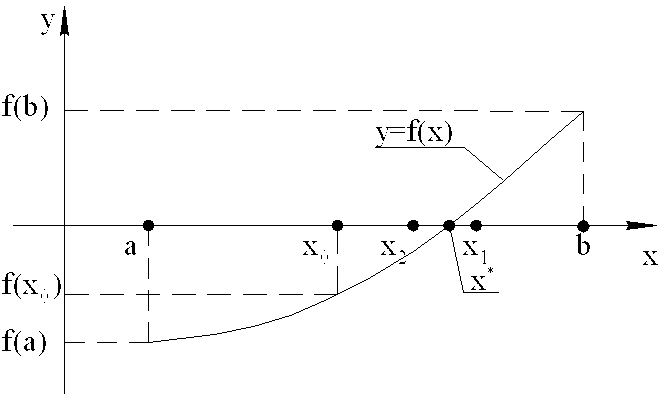


Рисунок 1.6 – Иллюстрация метода деления отрезка пополам.

2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДОВ

Для реализации методов вычисления корней нелинейных уравнений была разработана программа на языке программирования *С++,* способная точно находить необходимые значения аргумента.

Программа была реализована с использованием стандартных библиотек *C++* [5][6]: *iostream* для организации ввода-вывода данных и *cmath*, предоставляющей необходимый набор математических функций и констант. Применение директивы *using namespace std*; позволило упростить доступ к стандартным функциям и классам библиотеки, избегая излишней формальности в коде и повышая его читаемость.

Тело программы находится в основной функции *main,* которая является точкой входа в программу. Внутри функции происходят все основные действия для подготовки к нахождению корней.

При запуске программы пользователю первоначально предлагают выбрать исследуемую функцию из трёх предложенных (рисунок 2.1). Для этого необходимо ввести индекс (1, 2, 3) соответствующий каждой функции.

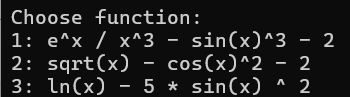


Рисунок 2.1 – Меню выбора исследуемой функции

Выбрав функцию, пользователю предложат выбрать метод, которым будет исследоваться функция (рисунок 2.2). Для этого необходимо ввести индекс, аналогично выбору функции.

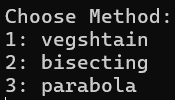


Рисунок 2.2 – Меню выбора метода исследования

Для удобства пользователя в программе предусмотрено меню, реализованное с помощью цикла *while* и оператора *switch*, которое будет представлено после вышеописанных действий (рисунок 2.3).

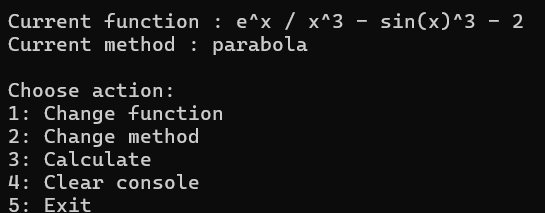


Рисунок 2.3 – Меню дальнейших действий

В меню отображены текущая функция и метод исследования, а так же предоставлен ряд действий, для выбора которых необходимо ввести индекс действия аналогично выбору функции и метода.

Действие *1: Change function* вызывает функцию *changeFunction*, которая вызывает меню выбора функции, идентичное рисунку 2.1.

Действие *2: Change method* вызывает функцию *changeMethod*, которая вызывает меню выбора метода, идентичное рисунку 2.2.

Действие *4:* *Clear console* вызывает функцию *system("cls")*, которая очищает весь текст в консоли.

Действия 1, 2 и 4 после выполнения снова вызывают меню дальнейших действий.

Действие *5: Exit* прерывает цикл, что приводит к полному завершению программы.

Действие *3: Calculate* запускает алгоритм подготовки к решению уравнения. Независимо от выбранной функции и метода пользователя попросят ввести левую и правую границу поиска – промежуток, содержащий корни уравнения, а так же количество сегментов, на которые этот промежуток будет разбит (рисунок 2.4).

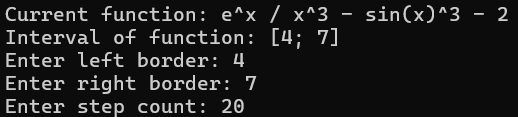


Рисунок 2.4 – Ввод границ промежутка и количества сегментов

Далее программа выводит таблицу значений *x*, соответствующие им значения функции *y* и индексы каждого из аргументов. Аргументы и их количество рассчитываются из ранее введённого промежутка и количества сегментов (рисунок 2.5).

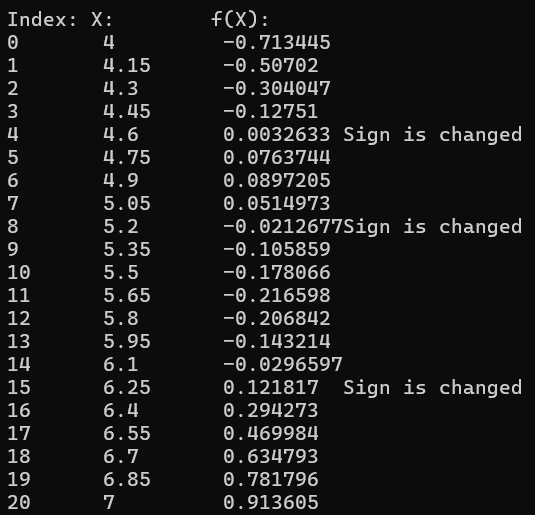


Рисунок 2.5 – Таблица значений

Пользователю необходимо выбрать индексы аргументов, которые будут являться левой и правой границей для получения точного значения корня уравнения. Для удобства в таблице так же выводится приписка о том, что функция поменяла свой знак по сравнению с предыдущим аргументом. Далее необходимо ввести необходимую точность и количество максимальных итераций (рисунок 2.6).

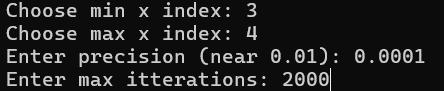


Рисунок 2.6 – Ввод параметров для вычисления

Программа перейдет к подсчету значения аргумента и выведет в консоль результат (рисунок 2.7).



Рисунок 2.7 – Вывод результата

2.1 Метод Вегстейна

Функция *vegshtain* является реализацией метода Вегстейна.

Функция принимает минимальное значение *x\_min* как левую границу, максимальное значение *x\_max* как правую границу, точность вычисления корня *precision*, количество максимальных итераций *itterationsCount* и исследуемую функцию *function*.

Принимаемые функцией значения присваиваются локальным переменным *x*\_0 *= x*\_*min, x*\_1 *= x*\_*max, eps = precision, k = itterationsCount*.

До начала поиска точных значений вычисляются значения функции *y\_0* и *y*\_1 в точках *x*\_0и *x*\_1 соответственно.

Далее запускается цикл с условием выхода при или же при . В теле цикла высчитывается значение *x* по формуле

Значение *x* из формулы (3.1) заменяет предыдущее значение *x*\_1 и высчитывается новое значение *y*\_1 уже с новым значением *x*\_1.

Алгоритм повторяется до тех пор, пока не будет найден корень с заданным приближением или пока количество итераций превысит лимит, что приведет к выходу из цикла.

2.2 Метод деления отрезка пополам

Функция *bisecting* является реализацией метода деления отрезка пополам.

Функция принимает минимальное значение *x*\_*min* как левую границу, максимальное значение *x*\_*max* как правую границу, точность вычисления корня *precision*, количество максимальных итераций *itterationsCount* и исследуемую функцию *function*.

Принимаемые функцией значения присваиваются локальным переменным *x*\_0 *= x\_min, x*\_1 *= x\_max, eps = precision, k = itterationsCount*.

До начала поиска точных значений вычисляются значения функции *y*\_0 и *y*\_1 в точках *x*\_0и *x*\_1 соответственно.

Далее запускается цикл с условием выхода при или же при . В теле цикла высчитывается значение *x* по формуле (3.2)

Далее значение функции *y* высчитывается в найденной точке . Знак значения функции *y* сравнивается со знаками значений *y*\_0 и *y*\_1 путем перемножения *y* на *y*\_0 и *y*\_1. В зависимости от знака полученного произведения присваиваются новые границы. При значениям *x* и *y* присваиваются значения *x*\_1и *y*\_1, при значениям *x* и *y* присваиваются значения *x*\_0и *y*\_0, если же ни одно из этих условий не выполнилось, то пересечений функции с осью абсцисс нет.

Алгоритм повторяется до тех пор, пока не будет найден корень с заданным приближением или пока количество итераций превысит лимит, что приведет к выходу из цикла.

2.3 Метод параболы

Функция *parabola* является реализацией метода параболы.

Функция принимает минимальное значение *x*\_*min* как левую границу, максимальное значение *x*\_*max* как правую границу, точность вычисления корня *precision*, количество максимальных итераций *itterationsCount* и исследуемую функцию *function*.

Принимаемые функцией значения присваиваются локальным переменным *a = x*\_*min, b = x*\_*max, eps = precision, k = itterationsCount*.

Далее запускается цикл с условием выхода при или же при . В теле цикла выполняется определенный алгоритм.

Суть этого алгоритма заключается в подсчете координат точек параболы, лежащих на кривой функции. В качестве опорных точек берутся крайние значения *a* и *b* а также их среднее арифметическое *c* = . После высчитываются значения функции , , в точках *a*, *b*, *c* соответственно.

На основе значений функции вычисляются точки пересечения параболы с осью *x* по формулам

где A, B и C высчитываются по формулам (3.5), (3.6) и (3.7) соответственно.

Далее из *x*\_0и *x*\_1 выбирается значение, которое находится в пределах интервала *a* и *b*, и присваивается переменной *x*\_*s*, предварительно сохранив её предыдущее значение в переменной *x*\_*p*. В точке *x*\_*s* вычисляется значение функции .

Значения функции сравнивается со знаками значений и , и значения *a* или *b* меняется на *x\_s* в соответствии со сменой знака или по сравнению с .

Алгоритм повторяется до тех пор, пока не будет найден корень с заданным приближением или пока количество итераций превысит лимит, что приведет к выходу из цикла.

3 АНАЛИЗ АЛГОРИТМОВ

3.1 Скорость алгоритмов

Рассмотрим время нахождения каждого из корней в зависимости от метода и типа уравнения. Для всех уравнений поставим одинаковые условия:

– интервал [*a*, *b*], содержащий три корня уравнения, разделен на 10 промежутков с фиксированным шагом ;

– границы поиска корня уравнения являются максимально приближенными к искомому значению в пределах 10 промежутков;

– приближение искомого корня составляет пять знаков после запятой;

– количество итераций не ограничено;

Для получения точных значений для каждого корня было проведено 10000 вычислений и взято от них среднее арифметическое. Результаты вычислений представлены в таблице 3.1.

Таблица 3.1 – Время вычисления корней уравнений

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | Время нахождения корня, мкс | | | | | | | | |
|  | | |  | | |  | | |
|  |  |  |  |  |  |  |  |  |
| Вегстейна | 1.93 | 2.23 | 1.66 | 2.68 | 2.67 | 3.70 | 1.77 | 2.07 | 1.93 |
| Параболы | 6.48 | 13.3 | 8.25 | 8.19 | 7.55 | 5.23 | 4.81 | 9.84 | 6.66 |
| Деления пополам | 8.95 | 17.9 | 11.3 | 6.82 | 7.48 | 5.20 | 6.22 | 6.81 | 6.36 |

Из таблицы 3.1 получаем таблицу 3.2 с средним временем нахождения корня для каждого метода.

Таблица 3.2 – Среднее время вычисления корней уравнений

|  |  |
| --- | --- |
| Метод | Среднее время нахождения корня, мкс |
| Вегстейна | 2.29 |
| Параболы | 6.90 |
| Деления пополам | 8.56 |

Из таблицы 3.2 можно сделать выводы, что метод Вегстейна является самым быстрым из представленных, а методы параболы и деления отрезка пополам в среднем уступают по скорости примерно в три и четыре раза соответственно.

3.2 Эффективность алгоритмов

Рассмотрим количество итераций для нахождения каждого из корней в зависимости от метода и типа уравнения. Для всех уравнений поставим одинаковые условия:

– интервал [*a*, *b*], содержащий три корня уравнения, разделен на 10 промежутков с фиксированным шагом ;

– границы поиска корня уравнения являются максимально приближенными к искомому значению в пределах 10 промежутков;

– приближение искомого корня составляет 10 знаков после запятой;

– количество итераций не ограничено;

Для получения точных значений для каждого корня было проведено 10000 вычислений и взято от них среднее арифметическое. Результаты вычислений представлены в таблице 3.3 и 3.4.

Таблица 3.3 – Количество необходимых итераций

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод | Количество итераций, мкс | | | | | | | | |
|  | | |  | | |  | | |
|  |  |  |  |  |  |  |  |  |
| Вегстейна | 6 | 6 | 5 | 5 | 6 | 5 | 6 | 7 | 7 |
| Параболы | 6 | 8 | 7 | 6 | 7 | 6 | 6 | 7 | 7 |
| Деления пополам | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 |

Из таблицы 3.3 получаем таблицу 3.4 с среднем количеством итераций каждого из методов.

Таблица 3.4 – Среднее количество необходимых итераций

|  |  |
| --- | --- |
| Метод | Среднее количество итераций |
| Вегстейна | 5.8 |
| Параболы | 6.6 |
| Деления пополам | 29 |

Из таблицы 3.4 можно сделать выводы, что метод Вегстейна обходит методы парабол и деления отрезка пополам. Разница между методом Вегстейна и методом параболы не существенна, в отличие от их разницы между методов деления отрезка пополам, который уступает примерно в 5 раз.

3.3 Потребление ресурсов

Рассмотрим занимаемую оперативную память. При вызове метода создается новый кадр стека (s*tack frame*), который помещается в стек вызова (call stack). В кадре стека содержится информация о функции, а также служебная информация.

Исходя из архитектуры функций, занимаемая память каждой из них будет состоять из следующих компонентов:

– пять передаваемых по ссылке параметров (40 байт);

– служебная информация стека вызова (16-32 байта);

– локальные переменные функции.

Конечные результаты представлены в таблице 3.5.

Таблица 3.5 – Занимаемая функциями оперативная память

|  |  |
| --- | --- |
| Функция | Занимаемая память, байт |
| *vegshtain* | 108 |
| *bisecting* | 108 |
| *parabola* | 180 |

Из таблицы 3.5 видно, что по сравнению с среднестатистическими современными компьютерами, занимаемая память ничтожно мала.

Рассмотрим загруженность центрального процессора. Проведя ряд тестов и исходя из экспериментальных данных в пунктах 3.1 и 3.2 можно сказать, что функции практически не потребляют ресурсов процессора из-за малого количества вычислений.

3.4 Общее сравнение

Рассмотрим данные из таблиц 3.2, 3.4 и 3.5.

Из таблицы 3.2 видно, что метод Вегстейна является самым быстрым, затрачивая в среднем всего 2.29 микросекунд на нахождение корня. Метод параболы, с результатом 6.90 микросекунд, занимает второе место, значительно уступая методу Вегстейна по скорости. Метод деления отрезка пополам показывает наихудший результат – 8.56 микросекунд.

Аналогичная тенденция наблюдается и в таблице 3.4, где метод Вегстейна требует наименьшего среднего количества итераций для нахождения корня – 5.8. Метод параболы незначительно уступает методу Вегстейна, требуя в среднем 6.6 итераций. Метод деления отрезка пополам снова оказывается менее эффективным, требуя в среднем 29 итераций для достижения результата.

Из-за низкого потребления ресурсов, можно сказать, что методы эквивалентны между собой, хотя при детальном рассмотрении метод параболы занимает почти в два раза больше памяти, однако из-за общего объема занимаемой памяти этой разницей можно принебречь.

Таким образом, метод Вегстейна демонстрирует наивысшую эффективность, значительно опережая остальные методы по обоим критериям. Метод параболы также показывает хорошие результаты, особенно по количеству необходимых итераций, но по скорости существенно уступает методу Вегстейна. Метод деления отрезка пополам, несмотря на свою простоту, значительно проигрывает в эффективности, требуя больше времени и итераций для нахождения корня.

3.5 Выводы

Каждый из рассмотренных методов решения нелинейных уравнений имеет свои особенности и области применимости. Рассмотрим их подробнее.

Метод Вегстейна, как и метод Ньютона, требует вычисления производных функции. Он полезен в случаях, когда требуется быстрая сходимость и когда известны хорошие начальные приближения. Часто применяется в задачах оптимизации и инженерных задачах, где требуется точность. Основными преимуществами метода являются быстрая сходимость и высокая эффективность в работе с функциями, имеющими простые производные. Недостатками могут являться необходимость нахождение производных и необходимость изначально точного приближения для максимальной эффективности работы алгоритма.

Метод парабол полезен в случаях, когда функция гладкая и есть возможность подобрать хорошие начальные точки. Часто применяется в задачах, где вычисление производных затруднительно или невозможно. Основными преимуществами метода являются быстрая сходимость при хороших начальных приближениях и отсутствие необходимости в производных. Недостатками могут являться необходимость выбора трех начальных точек и меньшая устойчивость к ошибкам в начальных приближениях.

Метод деления отрезка пополам полезен в случаях, когда требуется гарантированная сходимость и когда функция меняет знак на концах отрезка. Часто применяется в начальных этапах анализа функции для грубой локализации корней. Основными преимуществами метода являются простота реализации и гарантированная сходимость при наличии корня на интервале. Недостатками могут являться медленная сходимость и необходимость, чтобы функция меняла знак на концах начального интервала.

Выбор метода зависит от конкретных условий задачи, требуемой точности и доступных вычислительных мощностей.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе были рассмотрены три метода решения нелинейных уравнений: метод Вегстейна, метод деления отрезка пополам и метод параболы.

Теоретическая часть работы посвящена обзору основных понятий, связанных с нелинейными уравнениями, а также детальному описанию различных алгоритмов нахождения корней.

В практической части работы была разработана программа на языке программирования *C++* для решения нелинейных уравнений тремя необходимыми методами. Программа предоставляет гибкий и удобный интерфейс для решения уравнений.

В ходе работы были исследованы эффективность и скорость алгоритмов. В результате чего был получен вывод о том, что метод Вегстейна является наиболее эффективным во всех параметрах, метод парабол незначительно уступил методу Вегстейна, но также показал хорошие резулльтаты. Метод деления отрезка пополам оказался хоть и самым надежным, и простым, но от того и более медленным и неэффективным.

Таким образом, поставленная в работе задача – сравнение эффективности методов Вегстейна, параболы и деления отрезка пополам – была выполнена.

Результаты работы могут быть использованы для оптимизации вычислительных процессов в различных областях науки и техники, а так же для решения инженерных задач с необходимостью высокой точности.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Численные методы анализа [Электронный ресурс]: учебное пособие / Р.Ф. Ахвердиев, Ф.Г. Искакова, О.А. Слипченко. – Электрон. текстовые дан. (1 файл: 2,01 Мб). – Казань: Издательство Казанского университета, 2022. – 110 с. – Систем. требования: Adobe Acrobat Reader. – URL: https://kpfu.ru/portal/docs/F2035033661/\_red.\_\_ Posobie.po.chisl..met.pdf – Загл. с титул. Экрана

[2] Cиницын, А. К. Алгоритмы вычислительной математики : учебно-метод. пособие по курсу «Основы алгоритмизации и программирования» / А. К. Cиницын, А. А. Навроцкий. – Минск : БГУИР, 2007.

[3] Волков Е. Ф. Численные методы: Учеб. Пособие для вузов. – 2-е изд., испр. – М.: Наука. Гл. ред. физ.-мат. лит., 1987. ­– 248 с.

[4] Полянин А.Д. Методы решения нелинейных уравнений математической физики и механики / А.Д. Полянин, В.Ф. Зайцев, А.И. Журов – М.: ФИЗМАТЛИТ, 2005. – 256 с.

[5] Эффективный и современный С++: 42 рекомендации по исполыованию С++ 11 и С++14.: Пер. с англ. - М. : ООО "ИЛ. Вильяме", 2016. –304 с.: ил. – Пapал. тит. англ.

[6] Страуструп, Б.Б Язык программирования С++/ Б. Страуструп – М.: Бином, 2022. – 369 с.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Код программы**

*#include <iostream>*

*#include <iomanip>*

*#include <cmath>*

*using namespace std;*

*double functionV(double&);*

*double functionP(double&);*

*double functionD(double&);*

*int selectionMenu();*

*void changeFunction(int&);*

*void changeMethod(int&);*

*void vegshtain(double&, double&, double&, int&, double (\*)(double&));*

*void bisecting(double&, double&, double&, int&, double (\*)(double&));*

*void parabola(double&, double&, double&, int&, double (\*)(double&));*

*int main()*

*{*

*// массивы указателей на функции для удобства выбора*

*double(\*functionArray[])(double&) = { functionV, functionP, functionD };*

*void (\*methodArray[])(double&, double&, double&, int&, double (\*)(double&)) = { vegshtain, bisecting, parabola };*

*// массивы строк для удобного вывода*

*char functionIntervals[3][10] =*

*{*

*"[4; 7]",*

*"[4; 8]",*

*"[2; 6]"*

*};*

*char functionStrings[3][25] =*

*{*

*"e^x / x^3 - sin(x)^3 - 2",*

*"sqrt(x) - cos(x)^2 - 2",*

*"ln(x) - 6 \* sin(x) ^ 2"*

*};*

*char methodStrings[3][10] =*

*{*

*"vegshtain",*

*"bisecting",*

*"parabola"*

*};*

*// основные переменные, задающиеся условием*

*double leftBorder = 0; // a*

*double rightBorder = 0; // b*

*int functionStepCount = 0; // m*

*double functionStepValue = 0; // h*

*double eps; // eps*

*int itterationsCount; // it*

*// переменные для работы с аргументами*

*double\* values;*

*int index;*

*double currentFunctionValue;*

*// переменные для выбора границ поиска аргумента*

*int x\_min\_index;*

*int x\_max\_index;*

*// переменные для меню выбора*

*int menuIndex;*

*int functionTypeIndex;*

*int methodTypeIndex;*

*bool shouldWork = true;*

*// первоначальный выбор функции и метода*

*changeFunction(functionTypeIndex);*

*changeMethod(methodTypeIndex);*

*// цикл меню для удобного использования программы*

*while (shouldWork)*

*{*

*// вывод текущей информации*

*cout << endl << "Current function : " << functionStrings[functionTypeIndex] << "\nCurrent method : " << methodStrings[methodTypeIndex] << endl << endl;*

*// выбор действия*

*menuIndex = selectionMenu();*

*switch (menuIndex)*

*{*

*case 1:*

*changeFunction(functionTypeIndex);*

*break;*

*case 2:*

*changeMethod(methodTypeIndex);*

*break;*

*case 3:*

*#pragma region CALCULATION*

*// вывод/ввод требуемой информации*

*cout << endl << "Current function: " << functionStrings[functionTypeIndex] << endl;*

*cout << "Interval of function: " << functionIntervals[functionTypeIndex] << endl;*

*cout << "Enter left border: ";*

*cin >> leftBorder;*

*cout << "Enter right border: ";*

*cin >> rightBorder;*

*cout << "Enter step count: ";*

*cin >> functionStepCount;*

*// подсчет и вывод аргументов и значений функции при них в виде таблицы. Аргумент изменяется с определенным шагом*

*values = new double[functionStepCount] {0};*

*functionStepValue = abs(leftBorder - rightBorder) / functionStepCount;*

*cout << endl << left << setw(7) << "Index:" << setw(10) << "X:" << setw(10) << "f(X):" << endl;*

*index = 0;*

*for (double x = leftBorder; x <= rightBorder + functionStepValue / 2; x += functionStepValue)*

*{*

*currentFunctionValue = functionArray[functionTypeIndex](x);*

*cout << left << setw(7) << index << " " << setw(10) << x << setw(10) << currentFunctionValue << setw(15);*

*// вывод оповещения, что знак значения функции сменился*

*if (index != 0 && functionArray[functionTypeIndex](values[index - 1]) \* currentFunctionValue < 0)*

*cout << "Sign is changed" << endl;*

*else*

*cout << endl;*

*values[index++] = x;*

*}*

*// выбор минимального и максимального порога аргумента для более точного поиска*

*cout << "Choose min x index: ";*

*cin >> x\_min\_index;*

*cout << "Choose max x index: ";*

*cin >> x\_max\_index;*

*// ввод точности и количества иттераций*

*cout << "Enter difference min difference between x's (near 0.01): ";*

*cin >> eps;*

*cout << "Enter max itterations: ";*

*cin >> itterationsCount;*

*// вызов соответсвующей функции подсчета корня*

*methodArray[methodTypeIndex](values[x\_min\_index], values[x\_max\_index], eps, itterationsCount, functionArray[functionTypeIndex]);*

*break;*

*#pragma endregion*

*case 4:*

*// очистка консоли*

*system("cls");*

*break;*

*case 5:*

*// выход из цикла и завершение программы*

*shouldWork = false;*

*break;*

*}*

*}*

*return 0;*

*}*

*/// <summary>*

*/// функция смены функции для вычисления*

*/// </summary>*

*void changeFunction(int& type)*

*{*

*cout << "Choose function:\n1: e^x / x^3 - sin(x)^3 - 2\n2: sqrt(x) - cos(x)^2 - 2\n3: ln(x) - 5 \* sin(x) ^ 2" << endl;*

*cin >> type;*

*type--;*

*if (type < 0 || type > 2) type = 0;*

*}*

*/// <summary>*

*/// функция смены метода вычисления*

*/// </summary>*

*void changeMethod(int& method)*

*{*

*cout << "Choose Method:\n1: vegshtain\n2: bisecting\n3: parabola" << endl;*

*cin >> method;*

*method--;*

*if (method < 0 || method > 2) method = 0;*

*}*

*/// <summary>*

*/// функция выбора дальнейших действий*

*/// </summary>*

*int selectionMenu()*

*{*

*int action = 0;*

*cout << "Choose action:\n1: Change function\n2: Change method\n3: Calculate\n4: Clear console\n5: Exit" << endl;*

*cin >> action;*

*if (action <= 0 || action >= 6) action = 4;*

*return action;*

*}*

*/// <summary>*

*/// изначальная фунцкция для метода Вегстейна*

*/// </summary>*

*double functionV(double& x)*

*{*

*return exp(x) / pow(x, 3) - pow(sin(x), 3) - 2; //VEGSHTEIN*

*}*

*/// <summary>*

*/// изначальная фунцкция для метода Параболы*

*/// </summary>*

*double functionP(double& x)*

*{*

*return sqrt(x) - pow(cos(x), 2) - 2; //PARABOLA*

*}*

*/// <summary>*

*/// изначальная фунцкция для метода Деления отрезка пополам*

*/// </summary>*

*double functionD(double& x)*

*{*

*return log(x) - 5 \* pow(sin(x), 2); //DIVISION*

*}*

*/// <summary>*

*/// функция подсчета корня методом Вегстейна*

*/// </summary>*

*void vegshtain(double& x\_min, double& x\_max, double& precision, int& itterationsCount, double (\*function)(double&))*

*{*

*double x\_0 = x\_min;*

*double x\_1 = x\_max;*

*double eps = precision;*

*int k = itterationsCount;*

*double y\_0 = function(x\_0); // вычисляет базовое значние функции в минимальном аргументе*

*double y\_1 = function(x\_1); // вычисляет базовое значние функции в максимальном аргументе*

*double x = 0;*

*while (fabs(x\_1 - x\_0) > eps) // выполняем цикл, сравнивая разницу между аргументами*

*{*

*x = x\_1 - y\_1 \* (x\_1 - x\_0) / (y\_1 - y\_0); // сдвигает аргумент для следющего вычисления*

*x\_0 = x\_1; // заменяет первый аргумент на второй*

*x\_1 = x; // заменяет аргумент на новое потенциальное решение*

*y\_0 = y\_1; // заменяет первое значение функции на второе*

*y\_1 = function(x); // вычисляет новое второе значение функции с новым аргументом*

*// мы сравниваем, насколько близко значение функции к нулю с новыми x*

*// и с каждым шагом смотрим, находимся ли мы дальше или ближе к 0,*

*// и в зависимости от этого сдвигаем наш x на некоторое значение z*

*// выход из функции в случае превышения количества иттераций*

*if (--k == 0)*

*{*

*cout << "Cannot find any solutions. Out of irrotation" << endl;*

*return;*

*}*

*}*

*// выводим искомый аргумент с заданной точностью*

*cout << fixed << setprecision(static\_cast<int>(ceil(-log10(eps)))) << "Solution: x = " << x << endl;*

*}*

*/// <summary>*

*/// функция подсчета корня методом Деления отрезка пополам*

*/// </summary>*

*void bisecting(double& x\_min, double& x\_max, double& precision, int& itterationsCount, double (\*function)(double&))*

*{*

*double x\_0 = x\_min;*

*double x\_1 = x\_max;*

*double eps = precision;*

*int k = itterationsCount;*

*double y\_0 = function(x\_0); // вычисляет базовое значние функции в минимальном аргументе*

*double y\_1 = function(x\_1); // вычисляет базовое значние функции в максимальном аргументе*

*double x;*

*double y;*

*while (fabs(x\_1 - x\_0) > eps) // выполняем цикл, сравнивая разницу между аргументами*

*{*

*x = (x\_0 + x\_1) / 2; // вычисдяем аргумент между 1 и 2 аргументами*

*y = function(x); // вычисляем значение функции в среднем арументе*

*if (y\_0 \* y < 0) // перемещаем ПРАВУЮ границу к среднему значению, если функция пересекает ось X*

*{*

*x\_1 = x;*

*y\_1 = y;*

*}*

*else if (y\_1 \* y < 0) // перемещаем ЛЕВУЮ границу к среднему значению, если функция пересекает ось X*

*{*

*x\_0 = x;*

*y\_0 = y;*

*}*

*else // выходит из функции, если на этих отрезках нет пересечений с осью Х*

*{*

*cout << "There is no or more than 1 solution between " << x\_0 << " and " << x\_1 << endl;*

*return;*

*}*

*// выход из функции в случае превышения количества иттераций*

*if (--k == 0)*

*{*

*cout << "Cannot find any solutions. Out of irrotation" << endl;*

*return;*

*}*

*}*

*// выводим искомый аргумент с заданной точностью*

*cout << fixed << setprecision(static\_cast<int>(ceil(-log10(eps)))) << "Solution: x = " << (x\_0 + x\_1) / 2 << endl;*

*}*

*/// <summary>*

*/// функция подсчета корня методом Параболы*

*/// </summary>*

*void parabola(double& x\_min, double& x\_max, double& precision, int& itterationsCount, double (\*function)(double&))*

*{*

*double a = x\_min;*

*double b = x\_max;*

*double eps = precision;*

*int k = itterationsCount;*

*double x\_0;*

*double x\_1;*

*double c;*

*double x\_s = 0;*

*double x\_p = 0;*

*double f\_a, f\_b, f\_c, f\_x\_s;*

*double A, B, C;*

*do*

*{*

*c = (a + b) / 2; // вычисляем среднюю точку параболы*

*// вычисляем значения функций в точках параболы*

*f\_a = function(a);*

*f\_b = function(b);*

*f\_c = function(c);*

*// вычисляем специальные коэффициенты для дальнейших вычислений*

*A = (((f\_b - f\_c) / (b - c)) - ((f\_c - f\_a) / (c - a))) / (b - a);*

*B = ((f\_c - f\_a) / (c - a)) + A \* (a - c);*

*C = f\_a;*

*// вычисляем аргументы при помощи формулы*

*x\_0 = a - 2 \* C / (B + sqrt(B \* B - 4 \* A \* C));*

*x\_1 = a - 2 \* C / (B - sqrt(B \* B - 4 \* A \* C));*

*x\_p = x\_s;*

*// выбираем аргументы, находящиеся внутри определенных границ*

*if (a <= x\_0 && x\_0 <= b || a >= x\_0 && x\_0 >= b)*

*{*

*x\_s = x\_0;*

*}*

*else if (a <= x\_1 && x\_1 <= b || a >= x\_1 && x\_1 >= b)*

*{*

*x\_s = x\_1;*

*}*

*else*

*{*

*cout << "Cannot find any solutions" << endl;*

*return;*

*}*

*f\_x\_s = function(x\_s);*

*// выираем следующую границу, в зависимости от пересечения функцией оси Х*

*if (f\_a \* f\_x\_s < 0)*

*{*

*b = x\_s;*

*}*

*else if (f\_b \* f\_x\_s < 0)*

*{*

*a = x\_s;*

*}*

*// выход из функции в случае превышения количества иттераций*

*if (--k == 0)*

*{*

*cout << "Cannot find any solutions. Out of irrotation" << endl;*

*return;*

*}*

*} while (fabs(x\_p - x\_s) > eps); // выполняем цикл, сравнивая разницу между аргументами*

*// выводим искомый аргумент с заданной точностью*

*cout << fixed << setprecision(static\_cast<int>(ceil(-log10(eps)))) << "Solution: x = " << x\_s << endl;*

*}*