# Постановка задачи

Составить программу на Си, решающую (здесь и далее используется как синоним «находящую на заданном отрезке такое , что ») заданные в коде трансцендентные алгебраические уравнения численными методами.

* Найти с помощью метода Ньютона приближённое решение , лежащее на отрезке , и вывести его.
* Найти с помощью метода дихотомии приближённое решение , лежащее на отрезке , и вывести его.
* Найти с помощью метода итераций приближённое решение , лежащее на отрезке , и вывести его.
* Названные выше численные методы должны реализовываться в виде отдельных функций, принимающих на вход функцию-уравнение (далее – уравнение), левую и правую границы отрезка, содержащего корень, и точность вычислений. Некоторые функции могут иметь дополнительные аргументы, подобные производной в методе Ньютона.

На ввод программы подаётся точность вычислений, воспринимаемая как минимальное необходимое расстояние между двумя последними предложенными решениями.

Вывод программы (при вводе «1e-15») имеет следующий вид:

f1(0.54716075726033) = 0.0000000000000  
f2(1.07687398631180) = 0.0000000000000  
f3(1.23883997757415) = -0.0000000000000

# Организация исходного кода

|  |  |
| --- | --- |
| **Имя файла** | **Назначение** |
| k4.c | Управляющая функция, уравнения, ввод и вывод |
| helpers.h | Объявления вспомогательных функций |
| helpers.c | Реализация вспомогательных функций |
| root\_finding.h | Объявления функций, реализующих численные методы |
| root\_finding.c | Реализация численных методов |

Выбранная организация программы поясняется в третьей части курсового проекта. Повторим сказанное в ней:  
 «Как видно, программа поделена на файлы. Конечно, это может казаться лишним, но все-таки довольно полезно в ряде других случаев. Если рассматривать выпавшую программу, как частный случай, то да, можно делать из без разбиения на файлы, но в реальной жизни программистам часто требуется что-то менять, подстраивать под новые условия, и поэтому их интересует не столько, как работает функция, а что она в конечном итоге выдает. Поэтому, разбиение на файлы легко поможет программисту переделать программу за незначительное время из одного варианта в другой. Это, кстати, одна из причин появления в языке Си заголовочных файлов.

Для такой маленькой программы, наверное, и не стоит писать такое разбиение, ибо это может слегка затруднить программиста. Но если все сделать аккуратно и грамотно, то в конечном итоге поможет и организует работу. Поможет сделать программу более модифицируемой под иные схожие случаи».

Объявления и реализация численных методов выделены в отдельные файлы по тем же причинам: это довольно распространённые и общие операции, помещать которые в helpers, а тем более в k4, было бы неразумно.

## helpers.h, helpers.c[5]

|  |  |  |
| --- | --- | --- |
| **Функция** | **Аргументы** | **Возвращает** |
| epsilon(void) | - | double |
| factorial(int n) |  | double |
| power(double base, int exponent) |  | double |

Подробные разъяснения работы функций из helpers размещены в 3 части курсового проекта. Там же обсуждались важные для данной работы вопросы машинной арифметики, приближённых вычислений и погрешностей.

## root\_finding.h, root\_finding.c

В этих файлах содержатся реализации применяемых в программе численных методов в достаточно общем виде.

|  |  |  |
| --- | --- | --- |
| **Функция** | **Аргументы** | **Смысл** |
| double find\_root\_by\_bisection | double (\*f)(double x), double l, double r, double eps | Поиск корня функции f на отрезке с точностью eps |
| double find\_root\_by\_iterations | double (\*f\_iter)(double x), double l, double r, double eps | Поиск корня функции f на отрезке с точностью eps с применением функции f\_iter () |
| double find\_root\_by\_newton | double (\*f)(double x), double (\*f\_derivative)(double x), double l, double r, double eps | Поиск корня функции f (с использованием её производной f\_derivative) на отрезке с точностью eps |

Как обычно, более удобное и понятное объяснение расположено в файлах root\_finding.h или root\_finding.c.

## Метод дихотомии (find\_root\_by\_bisection(double (\*f)(double x), double l, double r, double eps))

Дихотомия применительно к программированию – понятие, порождающее довольно общий класс алгоритмов, реализующих поиск в упорядоченном наборе данных не за , а за .

В общих словах идея дихотомии хорошо иллюстрируется детской игрой «угадай число», в которой один игрок загадывает число от до , а второй пытается его угадать за минимальное число шагов. При абстрагировании от везения, интуиции и когнитивных эффектов, легко предложить оптимальную стратегию поиска числа . Поскольку для двух чисел можно однозначно сказать, что (, т.е. существует только 3 возможности при сравнении чисел, мы можем свести поиск к последовательному делению пополам области поиска.

Пусть .  
При делении области поиска пополам, лежит либо в правой половине области, либо в левой, либо равно разделителю.

. Следовательно, можно сузить область поиска в два раза (от 0 до 256), т.к. лежит слева от . Разделяя новую область, получаем середину . Поскольку , выбираем левую половину (от 0 до 128). Повторяя действие, замечаем, что теперь лежит справа от середины отрезка (). Теперь можно сказать, что лежит между 64 и 128. Находя середину данного отрезка , сужаем область до . Придерживаясь подобной стратегии, мы гарантированно найдём максимум за делений области поиска пополам.

Условия применимости метода дихотомии для поиска в некоторой области поиска таковы:

1. Любые два элемента области поиска сравнимы ().
2. Область поиска является отсортированной (в каком порядке – несущественно, но предположим, что по возрастанию), т.е. для области поиска .
3. .

Существуют некоторые особенности применимости метода, такие как требование дискретности и конечности области поиска (в противном случае можно говорить лишь о сходимости с некоторой точностью).

Замечу, что метод можно существенно обобщить, однако достигаемые при этом улучшения в понимании несопоставимы с затратами.

Для поиска корня уравнения условие, определяющее, справа или слева необходимо «закрепиться», таково: согласно теореме Больцано-Коши о промежуточных значениях, если функция имеет разные знаки на отрезке, то корень принадлежит ему. Следовательно, мы сокращаем область поиска до этого отрезка.

Таким образом, для задачи нахождения корня уравнения алгоритм имеет следующий вид:

*пока не достигнута требуемая точность:  
 середина = (левая граница + правая) / 2.0  
 если знак и различен:  
 правая граница = середина  
 иначе  
 левая граница = середина*

Очевидно, что если знаки функции в концах отрезка различны и функция непрерывная (иначе алгоритм может сойтись, к примеру, к вертикальной асимптоте гиперболы), то корень функции-уравнения лежит между ними.

Таким образом, это является достаточным условием сходимости метода, но не необходимым (чтобы убедиться, достаточно рассмотреть параболу на отрезке : , однако на существую корни ).

Замечу, что при заданной точности количество[4] вычислений функций будет в среднем около (подразумевается, что для одного аргумента функция вычисляется только один раз). Очевидно, для приведённой выше игры «угадай число» .

Убедимся, что непрерывна на и на концах отрезка имеет различные знаки.

Рассмотрим :



Проверим знаки на концах отрезка: , . Поскольку знаки различны и функция на отрезке непрерывна, заключаем, что метод дихотомии для нахождения корня применим.

## Метод итераций (find\_root\_by\_iterations(double (\*f\_iter)(double x), double l, double r, double eps))

Рассмотрим понятие неподвижной точки.

Пусть имеется отображение . Тогда называется множеством фиксированных точек.

Говоря о функциях, фиксированными точками называют решения уравнения .

Неподвижная точка называется притягивающей[2], если для любой достаточно близкой к ней точке последовательность стремится к .

Пусть . К примеру, . Тогда можно дать эквивалентное определение притягивающей точки :

для достаточно близкого к числа .

Учитывая геометрический смысл производной, можно заметить, что достаточным условием наличия притягивающей точки является на заданном отрезке.

Объединяя сказанное выше, кратко поясним метод итераций.

Пусть имеется непрерывная функция , для которой требуется найти нуль на отрезке . С помощью эквивалентных преобразований её можно привести к виду , причём для .

Тогда будет сходиться к нулю , если .

Эти же размышления можно записать в более привычном виде с использованием пределов.

Если последовательность сходящаяся, то . Тогда, полагая функцию непрерывной, перейдём к пределу в равенстве : или .

Таким образом, является нулём , равносильной .

Геометрическая интерпретация метода достаточно проста. Поскольку равносильно и метод приспособлен для работы с , будем пользоваться именно ей. Последовательность итераций зачастую выглядит как «лесенка» с постепенно уменьшающимся размером ступенек или как стремящаяся к своему центру угловатая спираль. Этот эффект «лесенки» - именно то, ради чего требовалось преобразовывать .

Итак, алгоритм вычисления корня с помощью метода итераций имеет примерно следующий вид:

*пока не достигнута требуемая точность:  
 x = F(x)  
вернуть x*

Найдём для функции : .

Отметим, что (очевидно, если заметить асимптоты и ), что говорит о монотонности . Поскольку знаменатель возрастает, а числитель – константа , монотонно убывает.

Следовательно, для проверки условия достаточно рассмотреть , которое истинно.

Доказательство непрерывности функции на отрезке полностью аналогично доказательству для .

## Метод Ньютона (find\_root\_by\_newton(double (\*f)(double x), double (\*f\_derivative)(double x), double l, double r, double eps))

Поскольку метод Ньютона – частный случай метода итераций (), опишем его кратко, затрагивая лишь выбор .

Пусть[3] на отрезке существует решение , а непрерывна и отлична от нуля (чтобы избежать деления на ноль) на . Тогда . Применяя формулу Лагранжа, получаем: . Производя замену , получаем . Выражая , получаем или, что равносильно , где - некоторое начальное приближение, в нашем случае равное .

Геометрический смысл[3]  метода Ньютона состоит в том, что для функции в точке начального приближения строится касательная, чьё пересечение с осью абсцисс берётся в качестве следующего приближения. Или, более формально, каждое приближение является абсциссой точки пересечения касательной, проведённой в к графику , с осью .

Модифицируя алгоритм метода итераций, получаем:

пока не достигнута требуемая точность:  
 *x = x – f(x)/f'(x)  
вернуть x*

Из графика и следует, что она и её производная непрерывны на . Поскольку производная функции монотонно возрастает, а , то на отрезке .

Можно проверить условие сходимости метода , однако решение данного неравенства настолько затратное и проблематичное, что проще проверить, является ли полученное с помощью метода Ньютона решение нулём функции. Как оказалось, предложенное методом Ньютона решение подходит под роль нуля функции.

Вычисление производной в точки можно было реализовать численно, однако поскольку даже с применением улучшений из [2] численное вычисление производной плохо работает вблизи нуля, оно не подходит для данной задачи, в которой основное действие происходит как раз вблизи нуля.

## k4.c

Данный файл содержит функции, задающие уравнение, и отвечает за ввод-вывод.

|  |  |  |
| --- | --- | --- |
| **Функция** | **Метод [решения]** | **Смысл/возвращает** |
| f1(double x) | Ньютона |  |
| f1\_derivative(double x) | Ньютона [производная] |  |
| f2(double x) | дихотомии |  |
| f3(double x) | итераций |  |
| main(void) | - | Ввод точность вычислений, вывод решений |

Алгоритм работы main тривиален:

*считать точность вычислений  
для каждой функции :  
 найти и вывести   
вернуть управление вызывающей программе[[1]](#endnote-1)*

# Выводы

Оказалось, что численные методы – очень занимательная и интересная научная область, кишащая тонкостями и ухищрениями как из математики, так и из информатики.

В процессе выполнения работы было исследовано и изучено множество интересных моментов, связанных с теоретическими и практическими аспектами численных методов, а также их применимостью.

В процессе анализа задачи стало как никогда ясно, сколь плодотворный симбиоз для обеих сторон порождает совместное использование математики и средств информатики.

Как и ранее, ценным оказался и приобретённый опыт разработки алгоритмов и написания кода, а также форсированное повышение уровня понимания математического анализа и его междисциплинарных связей.

# Использованные источники

1. 1 Математический анализ ч.1. – М.: МЦНМО, Зорич В.А, 2002

   2 Numerical Recipes in C: The Art of Scientific Computing, William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, 1992

   3 Википедия, http://ru.wikipedia.org /  
    [↑](#endnote-ref-1)