Ворожцов Артём Викторович
Кандидат физико-математических наук,
преподаватель кафедры информатики
Московского физико-технического института (МФТИ),
тренер сборной команды МФТИ по программированию.

Поиск корня уравнения методом деления пополам

Задача. Рассмотрим уравнение

$$e^x = x+2,\ e = 2{,}71828...$$

Точки пересечения графиков функций $f_1(x) = e^x$ и $f_2(x) = x+2$ являются корнями этого уравнения. Из графиков видно, что корней у данного уравнения два. Как узнать, чему они равны?

Сразу скажем, что никакой простой формулы, по которой можно было бы определить значение этих корней, нет. Математики в таких случаях говорят, что корни не выразимы через элементарные функции от целых чисел. Другими словами, нельзя, используя целые числа, операции деления, умножения, сложения, а также функции корня, квадрата, степени, экспоненты, логарифма, синуса, арксинуса ... и другие изучаемые в школе функции, получить точное выражение для корней этого уравнения.

Но ничто не мешает нам находить численные значения корней со сколь угодно большой точностью.

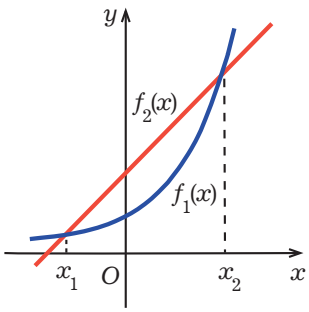

Рис. 1. Графики функций $f_1(x) = e^x$ и $f_2(x) = x+2$

Задача. С помощью компьютера необходимо найти положительный корень данного уравнения с погрешностью не более 10^{-10}.

Один из простейших алгоритмов вычисления корня основан на **методе деления пополам.**

Пусть нам требуется найти с некоторой точностью корень уравнения $f(x)=0$, который находится на промежутке $[l, r)$. Предположим также, что функция $f(x)$ непрерывна и в

Табл. 1

Сравнение реализаций метода деления пополам.

```
// Рекурсивная реализация
// алгоритма.
double root(double l, double r)
{
  double c = (l + r) / 2;
  if (r - l < EPS) return c;
  if ( f(c) * f(r) <= 0 )
    return root(c,r);
  else
    return root(l,c);
}
```

```
// Реализация, основанная
// на итерациях.
double root(double l, double r)
{
  double c = (l + r) / 2;
  while ( r - l > EPS ) {
    if (f(c) * f(r) <= 0)
      l = c;
    else
      r = c;
    c = ( l + r ) / 2;
  }
  return c;
}
```

точках l и r принимает значения разных знаков. Мы можем поступить следующим образом. Найдём середину c промежутка $[l, r)$. Корень находится на одном из полуинтервалов: $[l, c)$ или $[c, r)$. Выберем нужный из двух полуинтервалов и применим к нему те же рассуждения (см. рис. 2). Будем продолжать такое деление пополам, пока размер полуинтервала не станет меньше необходимой точности. Алгоритм поиска корня уравнения методом деления пополам основан на простой рекурсивной идее: **задача поиска корня на промежутке $[l, r)$ сводится к задаче поиска корня на $[c, r)$ или на $[l, c)$, где $c = (l+r)/2$.** Но эту идею можно реализовать, не используя рекурсивные функции. Каждый раз, рассматривая отрезок, мы переходим только к одной из его половин. Это можно осуществить в цикле, обновив значения l или r (см. табл. 1).

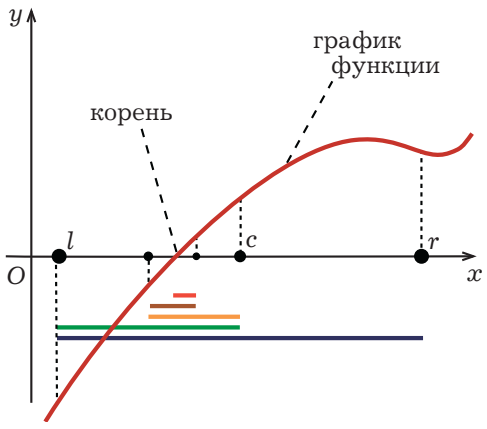

Рис. 2.

Вот полное простое решение поставленной задачи на Си:

```
/* Вычисление корня
трансцендентного уравнения*/
#include <stdio.h>
#include <math.h>
#define EPS 1e-10
double f ( double x ) {
  return exp(x) - 2 - x;}
int main() {
  double l = 0, r = 2, c;
  while ( r - l > EPS ) {
   c = ( l + r ) / 2;
   if (f(c) * f(r) <= 0)
     l = c;
   else
     r = c;
  }
printf("%.10lf\n",(l+r)/2);
}
```

При компиляции этой программы с помощью компилятора **GNU** (**GNU Compiler**) следует указать опцию **-lm**, которая указывает, что при компоновке программы необходимо подключить библиотеку **libm** с математическими функциями:

```
> gcc -lm root.c -o root
```

В библиотеке **libm**, в частности, определена функция **exp**, вычисляющая экспоненту, а также многие другие математические функции: тригонометричекие (**sin**, **cos**, **asin**, **acos**, **tan**, …), корень (**sqrt**), степень (**pow**), логарифм (**log**), …

Директива **#define EPS 1e-10** означает: в тексте программы идентификатор **EPS** заменить на число **1e-10**, то есть $1 \cdot 10^{-10}$ (в языке **Pascal** для этой цели используются блок **Const**). Число **EPS** — это погрешность, с которой мы хотим найти корень.

В рассмотренном нами примере вычисляется нуль вполне конкретной функции: **f(x)=exp(x)-2-x**. Следуя идеологии ***code reuse*** (**повторное использование кода**), полезно сделать функцию, которая может находить нули произвольной данной функции. Для этого нужно научиться передавать функцию в качестве аргумента. Это возможно и совсем несложно:

```
/*Универсальная функция вычис-
ления корня ур-ния f(x)=0 */
#include <stdio.h>
#include <math.h>
#define EPS 1e-16
double root(double l,double r,
        double (*f)(double)) {
  double c;
  while( r - l > EPS ) {
   c = ( l + r ) / 2;
   if( f(c) * f(r) <= 0 )
     l = c;
   else
     r = c;
  }
  return l;
}
double f1(double x)
      {return cos(x) - 3 * x;}
double f2(double x)
      {return exp(x) - x - 2;}
int main() {
// вычисляем и выводим корень
// уравнения f1(x) = 0
  printf("root1 = %lf\n",
        root(0, 2, f1));
// вычисляем и выводим корень
// уравнения f2(x) = 0
  printf("root2 = %lf\n",
        root(0, 2, f2));
  return 0;
}
```

Вывод этой программы выглядит так:

```
root1 = 0.316751
root2 = 1.146193
```

Вопросы и задачи

- За один шаг длина промежутка $[l, r)$ уменьшается в два раза, за три шага — в 8 раз. Сколько нужно шагов, чтобы уменьшить этот промежуток более, чем в 1000 раз?

- Сколько требуется шагов, чтобы, начиная с отрезка длины 2, дойти до отрезка длины меньше 10^{-10} ?

- Сколько требуется шагов, чтобы найти корень с точностью до 100 знаков после запятой?

- В случае деления пополам у нас есть нижняя и верхняя граница для значения корня. С каждым шагом эти границы сближаются. В методе Ньютона нахождения корня уравнения у нас имеется одно число x — текущее приближение корня. И следующее приближение получается по следующему алгоритму: находим точку на графике с абциссой x и проводим из неё касательную к графику; абсцисса точки пересечения касательной с осью абцисс будет новым значением x. Этот шаг повторяется, пока новое x отличается от старого более, чем на $10^{-10}/2$. Реализуйте этот алгоритм на каком-либо языке программирования. Для этого вам понадобится определить ещё одну функцию

$$f'(x) = (e^x - x - 2)' = e^x - 1,$$

которая возвращает значение производной. Убедитесь, что в этом алгоритме число итераций существенно меньше, нежели в алгоритме вычисления методом деления пополам. Если в методе деления пополам, чтобы получить очередную цифру после запятой, нужно проделать примерно три итерации, то в этом алгоритме каждая итерация примерно удваивает число верных цифр после запятой.

- Напишите программу, которая вычисляет максимальный корень уравнения $\cos(x) = x/100$.