Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.01 – «Информатика и вычислительная техника»

**Лабораторная работа №1**

**“Решение нелинейных уравнений”**

**Вариант 19.**

|  |
| --- |
| Выполнила студентка гр. ИВТ-24-2б  Косиненко Ксения Николаевна \_\_\_\_\_\_ |
| Проверил:  Доц. Каф. ИТАС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Ольга Андреевна Полякова\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка) (подпись)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

г. Пермь, 2024

**Метод итераций**

**Постановка задачи**

Решить уравнение  методом итераций. Уравнение передать в функцию как параметр с помощью указателя.

**Геометрическая интерпретация**

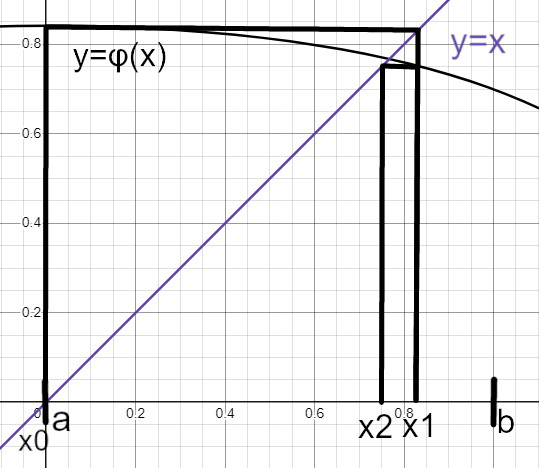
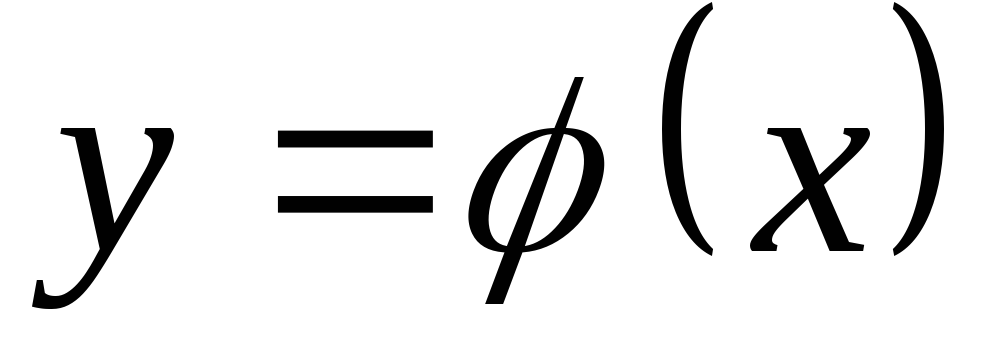
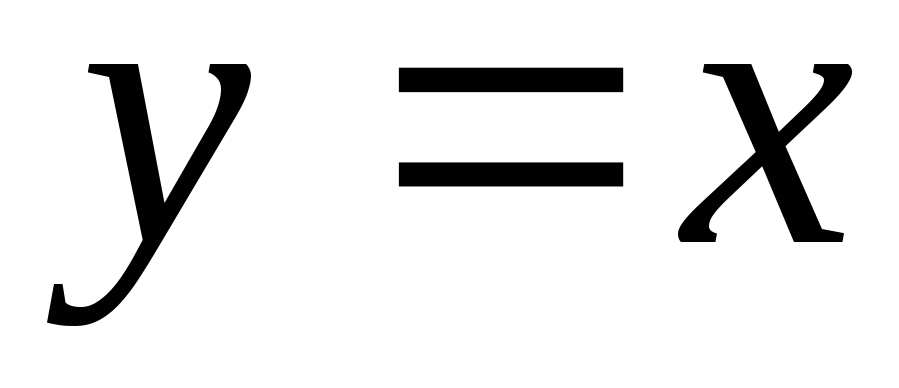
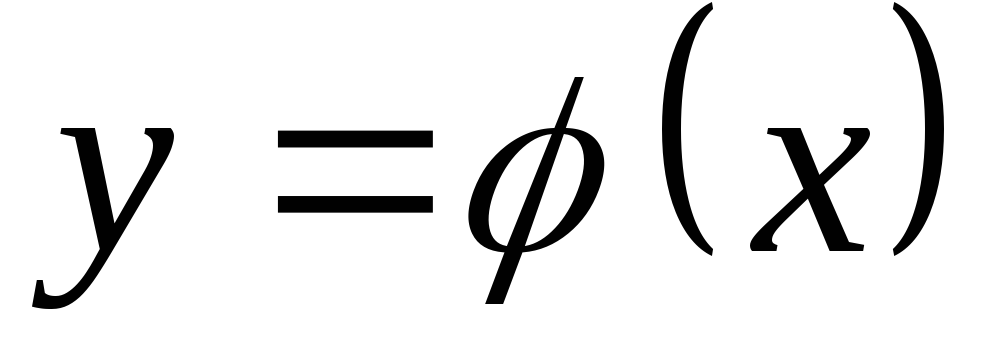
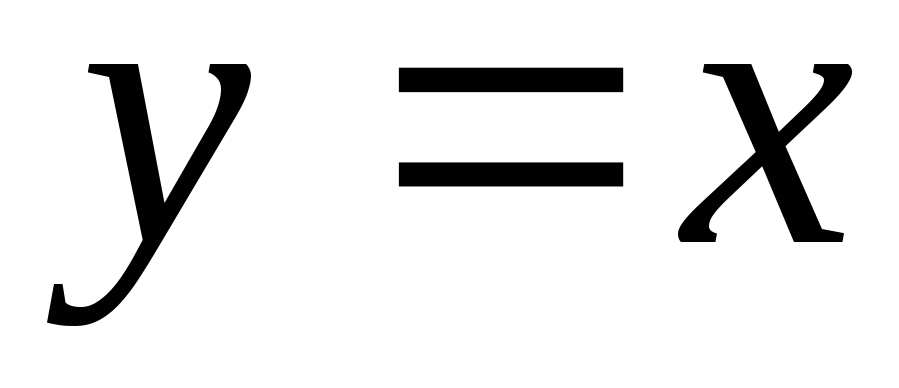
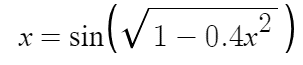
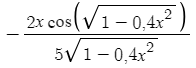


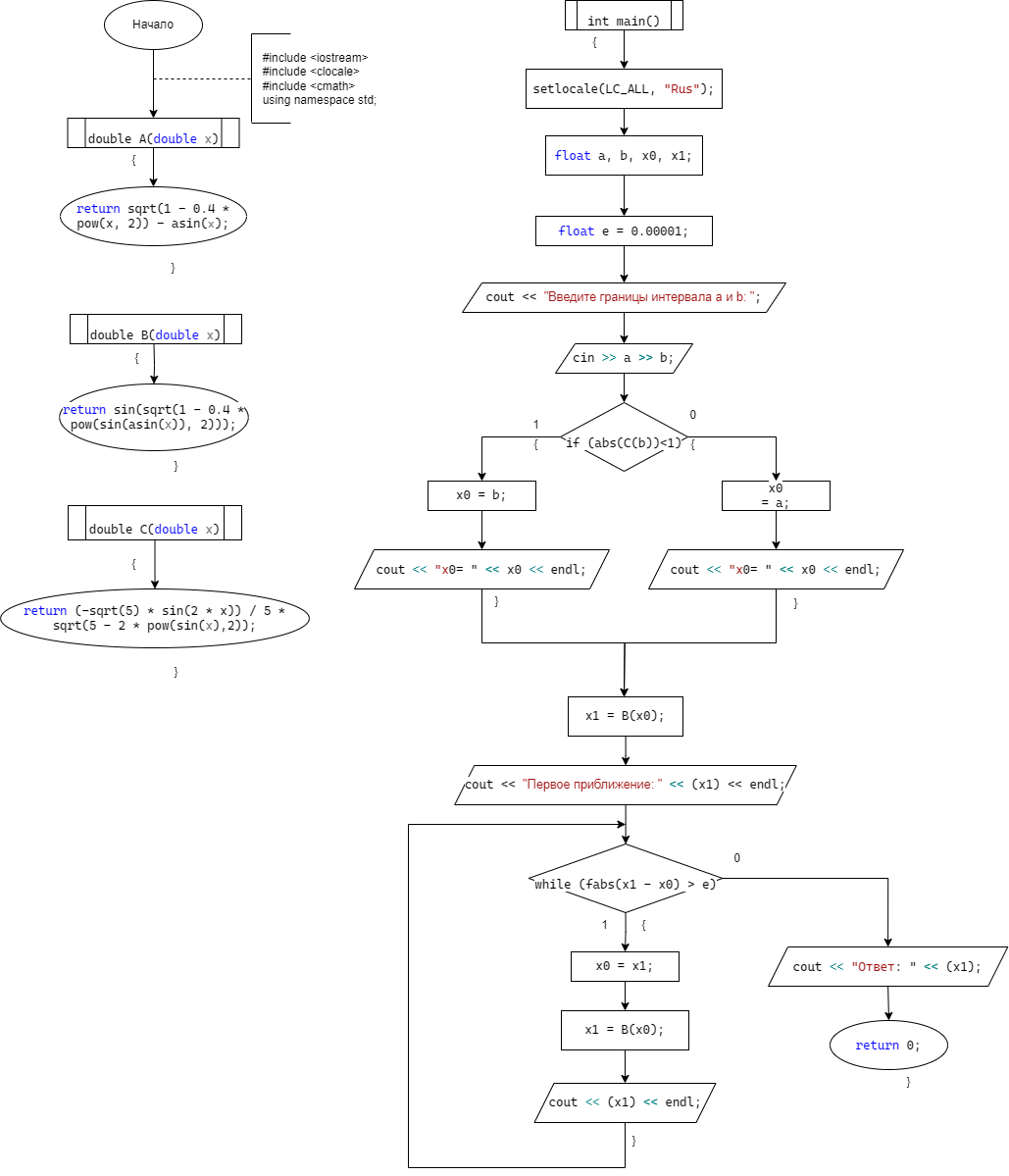
Рисунок 1. - Геометрическая интерпретация метода итераций.

Рисунок 1 иллюстрирует геометрический смысл метода простой итерации. Для того чтобы, имея точку, построить точку, необходимо из точкивосставить перпендикуляр к оси *Ох* до пересечения с графиком функции (см. Вывод формулы нахождения корня), провести через эту точку прямую параллельную оси *Ох* до пересечения с прямой  и опустить из этой точки перпендикуляр на ось *Ох*. В основании последнего перпендикуляра получим точку  . Таким образом, последующие корни сходятся к точке пересечения  и , которая и является корнем уравнения.

**Вывод формулы нахождения корня**

1. Дана функция y=. Корень на интервале ab = [0;1]
2. Запишем функцию .
3. Выразим x. =>  =φ(x) .
4. Выбираем сторону подхода к функции: Если |φ '(a)|<1, то условия выполнимости метода выполняются в точке a, x0=a; если |φ '(b)|<1, то условия выполнимости метода выполняются в точке b, x0=b. Т.к. φ '(x) = , то |φ '(a)| = 0(<1), а |φ '(b)| = 0,184536(<1), следовательно сторона подхода произвольная, выберем a.
5. Производим поиск корня xn+1= φ(xn), до тех пор когда |x1-x0|<=Ɛ, где Ɛ – заданная точность вычисления корня.

**Блок-схема со вписанным кодом**



**Программный код**

#include <iostream>

#include <clocale>

#include <cmath>

using namespace std;

double A(double x)

{

return sqrt(1 - 0.4 \* pow(x, 2)) - asin(x);

}

double B(double x)

{

return sin(sqrt(1 - 0.4 \* pow(sin(asin(x)), 2)));

}

double C(double x)

{

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

}

int main()

{

setlocale(LC\_ALL, "Rus");

float a, b, x0, x1;

float e = 0.00001;

cout << "Введите границы интервала а и b: ";

cin >> a >> b;

if (abs(C(b))<1)

{

x0 = b;

cout << "х0= " << x0 << endl;

}

else

{

x0 = a;

cout << "х0= " << x0 << endl;

}

x1 = B(x0);

cout << "Первое приближение: " << (x1) << endl;

while (fabs(x1 - x0) > e)

{

x0 = x1;

x1 = B(x0);

cout << (x1) << endl;

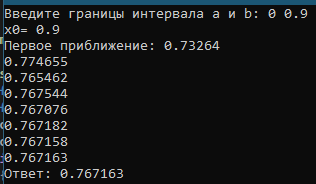
}

cout << "Ответ: " << (x1);

return 0;

}

Вывод:



**Метод Ньютона**

**Постановка задачи**

Решить уравнение  методом Ньютона (методом касательных). Уравнение передать в функцию как параметр с помощью указателя.

**Геометрическая интерпретация**

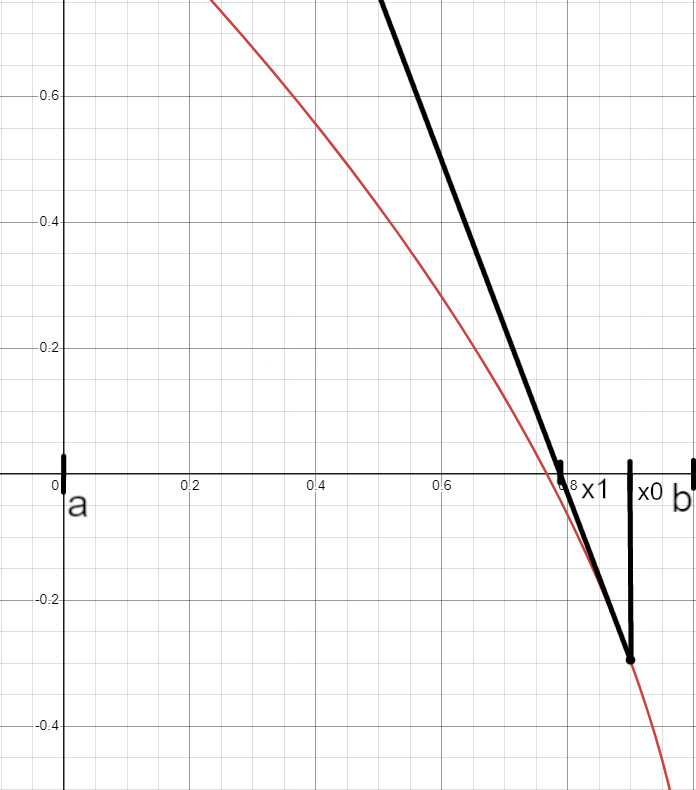


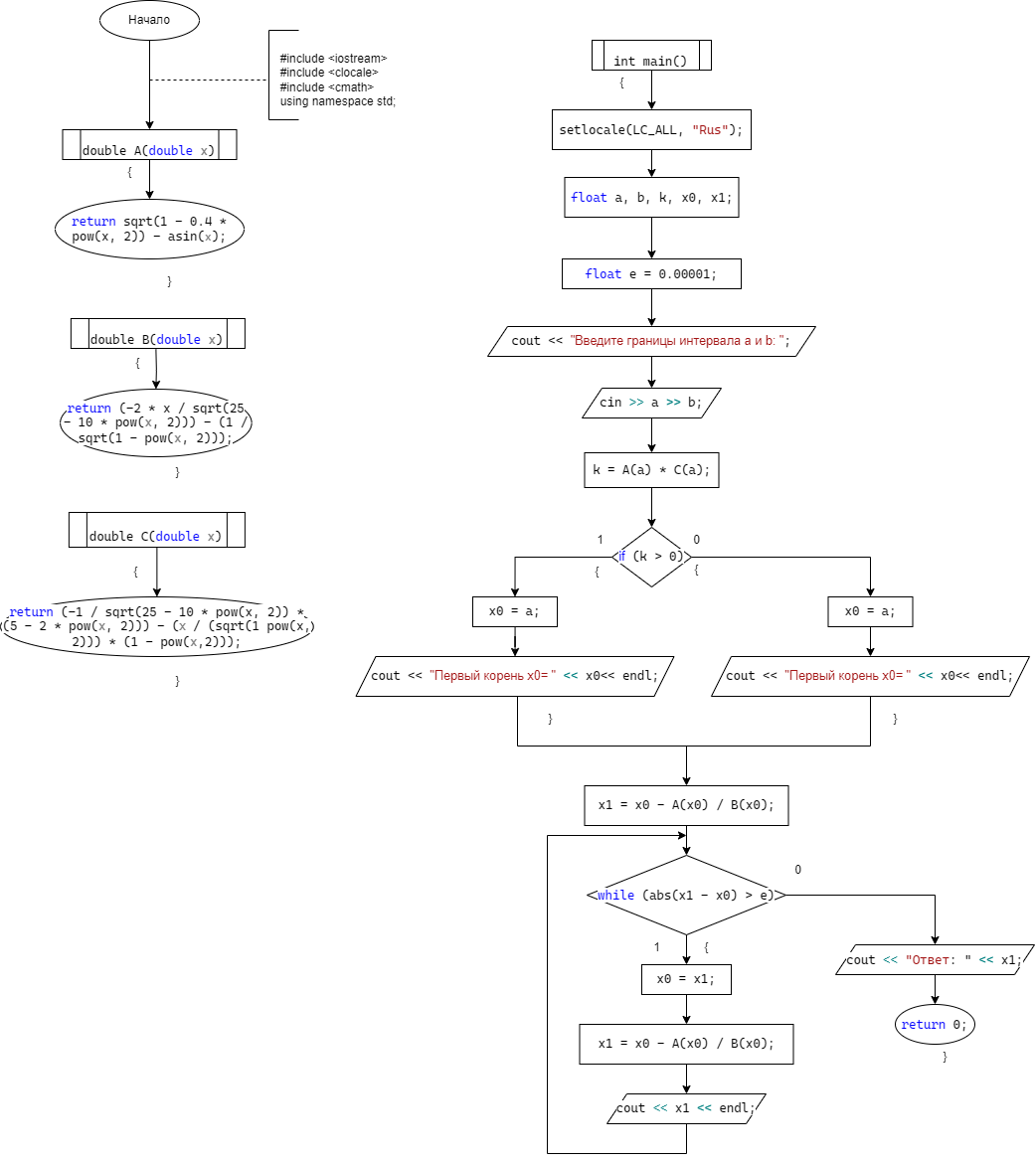
Рисунок 2 - Геометрическая интерпретация метода Ньютона.

Рисунок 2 иллюстрирует геометрический смысл метода Ньютона. Для того чтобы, имея точку, построить точку, необходимо из точки опустить перпендикуляр к оси *Ox* и через точку пересечения перпендикуляра с графиком провести касательную к графику функции. Точка  будет лежать на пересечении касательной с осью *Ox.* Таким образом, последующие корни сходятся к точке пересечения графика функции и оси *Ox*, которая и является корнем уравнения.

**Вывод формулы нахождения корня**

1. Дана функция y=. Корень на интервале ab = [0;1]
2. Угол касательной к функции f(x) определяется тангенсом угла наклона касательной к *Ox* через f'(x). f(x0) = tg(a) = k.
3. Т.к. касательная - это прямая, то запишем её уравнение в виде y=kx+b.
4. Выбираем сторону подхода к функции: Т.к. f'' показывает выпуклость или вогнутость функции, то если f(a) \* f''(a)>0, необходимо идти, выбирая x0 от границы a; если f(b) \* f''(b)>0, необходимо идти, выбирая x0 от границы b. f ''(x) = . f(a) \* f''(a)= -0.4 (<0), f(b) \* f''(b) = -∞\*(-0,7) (>0). Следовательно, подходить нужно из точки b.
5. Запишем уравнение касательной в x0. f(x0) =f '(x0) \* x+b.
6. Выразим b. b=f(x0)-f '(x0) \* x0
7. Подставим выражение 4 в п. 3. y=f '(x0) \* x0 + f(x0) – f '(x0) \* x0.
8. Вынесем общий множитель. y=f '(x0) \* (x-x0) + f(x0)
9. f '(x0) \* (x-x0) + f(x0) = 0
10. x1=x0-f(x0)/f '(x1)

**Блок-схема со вписанным кодом**



**Программный код**

#include <iostream>

#include <clocale>

#include <cmath>

using namespace std;

double A(double x)

{

return sqrt(1 - 0.4 \* pow(x, 2)) - asin(x);

}

double B(double x)

{

return (-2 \* x / sqrt(25 - 10 \* pow(x, 2))) - (1 / sqrt(1 - pow(x, 2)));

}

double C(double x)

{

return (-1 / sqrt(25 - 10 \* pow(x, 2)) \* (5 - 2 \* pow(x, 2))) - (x / (sqrt(1 - pow(x, 2))) \* (1 - pow(x, 2)));

}

int main()

{

setlocale(LC\_ALL, "Rus");

float a, b, k, x0, x1;

float e = 0.0001;

cout << "Введите границы интервала а и b: ";

cin >> a >> b;

k = A(a) \* C(a);

if (k > 0)

{

x0 = a;

cout << "Первый корень х0= " << x0 << endl;

}

else

{

x0 = b;

cout << "Первый корень х0= " << x0 << endl;

}

x1 = x0 - A(x0) / B(x0);

while (abs(x1 - x0) > e)

{

x0 = x1;

x1 = x0 - A(x0) / B(x0);

cout << x1 << endl;

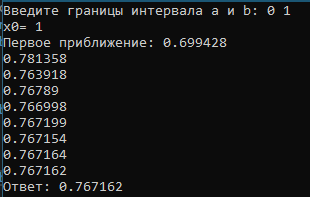
}

cout << "Ответ: " << x1;

return 0;

}

Вывод:

 **Метод половинного деления**

**Постановка задачи**

Решить уравнение  методом половинного деления (методом дихотомии). Уравнение передать в функцию как параметр с помощью указателя.

**Геометрическая интерпретация**

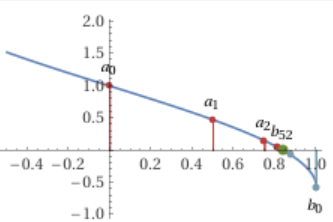
****

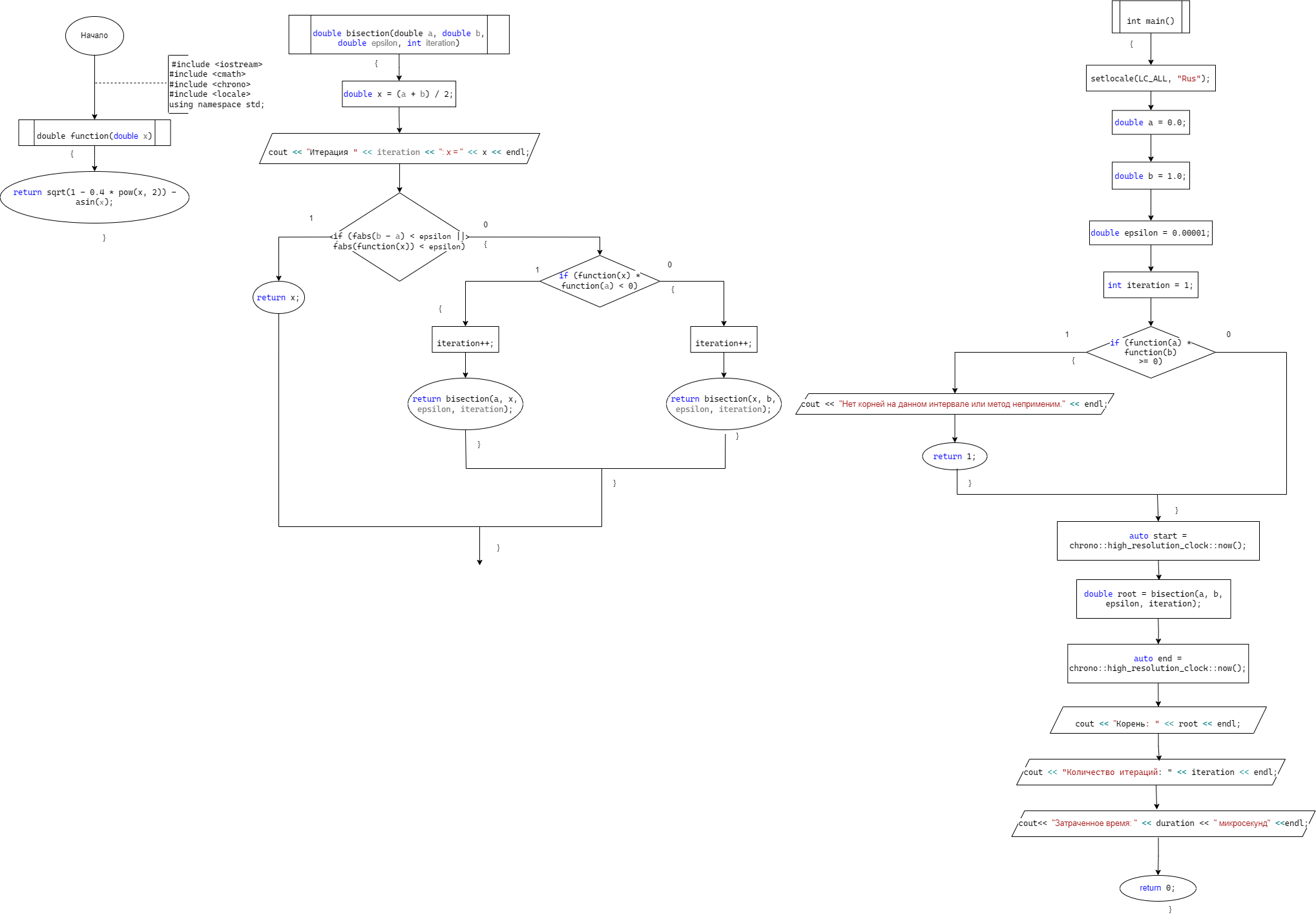
Рисунок 3 - Геометрическая интерпретация метода половинного деления.

Рисунок 3 иллюстрирует геометрический смысл метода половинного деления. Для того чтобы, имея точку, построить точку, необходимо на начальном этапе выбрать отрезок [a,b], на котором функция f(x) меняет знак, что указывает на наличие корня внутри этого интервала. Затем находится середина интервала , и проверяется знак функции в средней точке. Если знак функции в средней точке противоположен знаку функции на одном из концов отрезка, корень находится в соответствующей половине отрезка. Процесс повторяется, каждый раз сужая интервал локализации корня.

**Вывод формулы нахождения корня**

1. Дана функция y=. Корень на интервале ab = [0;1]
2. Середина интервала =0,5.
3. Вычисляем значения функции в точках a, c и b. f(a)=f(0)=1−0,4\*02−arcsin(0)=1−0=1, f(c)=f(0,5)=1−0,4\*0,52−arcsin(0,5), f(b)=f(1)=1−0,4\*12−arcsin(1)
4. Определение нового интервала. Если f(0) \*f(0,5) <0, корень находится в левой половине интервала [0;0,5]. Если f(0,5) \*f(1) <0, корень находится в правой половине интервала [0,5;1].
5. Процесс повторяется до тех пор, пока длина интервала не станет меньше заданной точности ε.

**Блок-схема со вписанным кодом**



**Программный код**

#include <iostream>

#include <cmath>

#include <chrono>

#include <locale>

using namespace std;

double function(double x)

{

return sqrt(1 - 0.4 \* pow(x, 2)) - asin(x);

}

double bisection(double a, double b, double epsilon, int iteration)

{

double x = (a + b) / 2;

cout << "Итерация " << iteration << ": x = " << x << endl;

if (fabs(b - a) < epsilon || fabs(function(x)) < epsilon)

{

return x;

}

else if (function(x) \* function(a) < 0)

{

iteration++;

return bisection(a, x, epsilon, iteration);

}

else

{

iteration++;

return bisection(x, b, epsilon, iteration);

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

double a = 0.0;

double b = 1.0;

double epsilon = 0.00001;

int iteration = 1;

if (function(a) \* function(b) >= 0)

{

cout << "Нет корней на данном интервале или метод неприменим." << endl;

return 1;

}

auto start = chrono::high\_resolution\_clock::now();

double root = bisection(a, b, epsilon, iteration);

auto end = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(end - start).count();

cout << "Корень: " << root << endl;

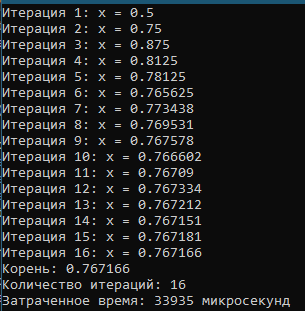
cout << "Количество итераций: " << iteration << endl;

cout << "Затраченное время: " << duration << " микросекунд" << endl;

return 0;

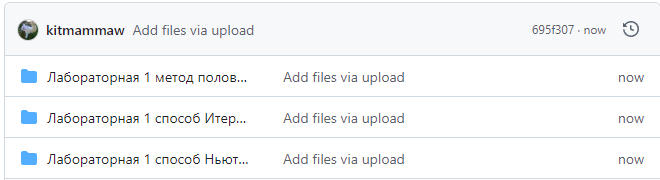
}

Вывод:



**GitHub**

<https://github.com/kitmammaw/->

****