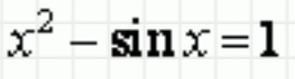
**Отчет курсанта Громова Григория Андреевича группы 22.Б05 о выполнении практического задания на 10.11.2022**

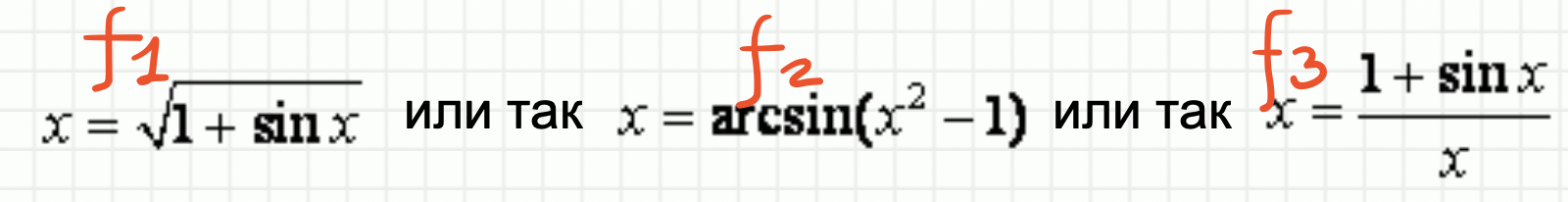
1. **Метод итерации**

Алгоритм:

1. Из данного уравнения выразим x несколькими способами, получив в итоге функции вида x = f(x)
2. Задаем первое приближения, проведя математический анализ поведения функции (определить непрерывные участки, найти экстремумы и определить монотонность, и, например, по теореме Больцано-Коши сделать вывод о промежутке, на котором находится корень)
3. Вводим коэффициент погрешности, меняя который, мы будем менять точность значения x
4. Пока x отличается по модулю от своего предыдущего значения на величину, меньшую введенной нами в пункте 4, вычисляем новые значения x по формуле из пункта 1

Рассмотрим работу программы на примере уравнения 

Тогда x можно выразить, например, так:



Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double f1(double x);

double f2(double x);

double f3(double x);

int main(){

double x = 1; //первое приближение

double x\_pred; //переменная, куда мы будем записывать предыдущее значение x

double dx = 9999; //насколько отличается x от своего прошлого значения (изначально ставим большое число

//чтобы точно произошел вход в цикл, а далее ее значение вычисляется непосредственно)

double pogresh = 0.00001; //вводим коэффициент погрешности

int counter = 0; //счетчик количества итераций

while (fabs(dx) > pogresh){ //пока число x n-ное отстоит от x (n-1)-ное не менее, чем на dx

x\_pred = x;

x = f1(x\_pred); //вычисляем следующее значение x ("следующий x")

//можно поменять на "x = f2(x\_pred)" или "x = f3(x\_pred)" в зависимости от того,

//как мы выразили x

dx = x - x\_pred; //вычисляем насколько оно отстоит от предыдущего

counter++;

printf("x\_pred = %lf, x = %lf, dx = %lf\n", x\_pred, x, dx); //промежуточные значения переменных

}

printf("x = %lf (вычислено за %d итераций)", x, counter); //выводим результат

return 0;

}

double f1(double x){ //первый способ выражения x

return pow(1 + sin(x), 0.5);

}

double f2(double x){ //втрой способ выражения x

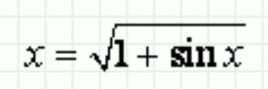
return asin(x\*x - 1);

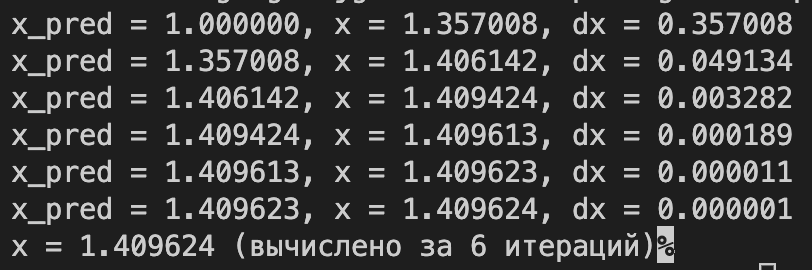
}

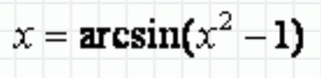
double f3(double x){ //третий способ выражения x

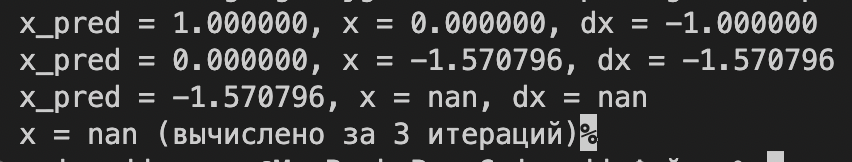
return (1 +sin(x) / x);

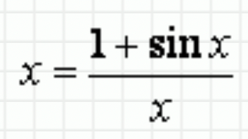
}

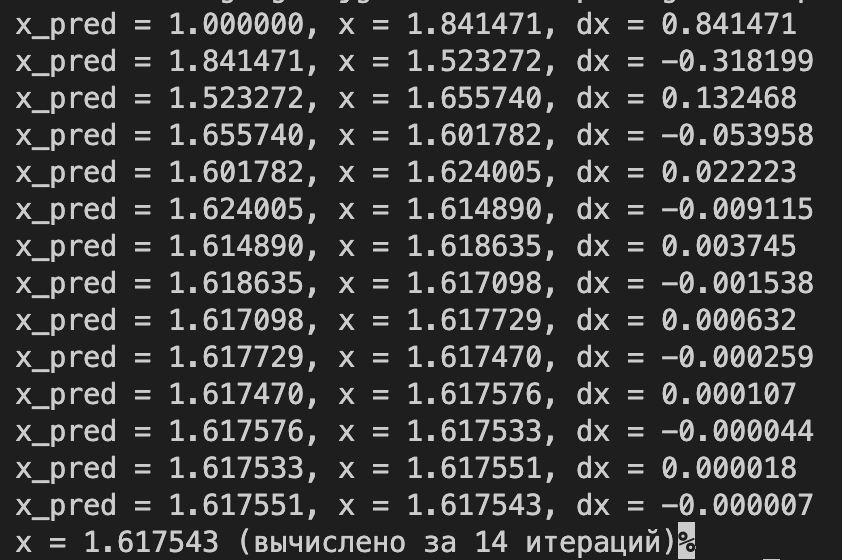
При выборe :



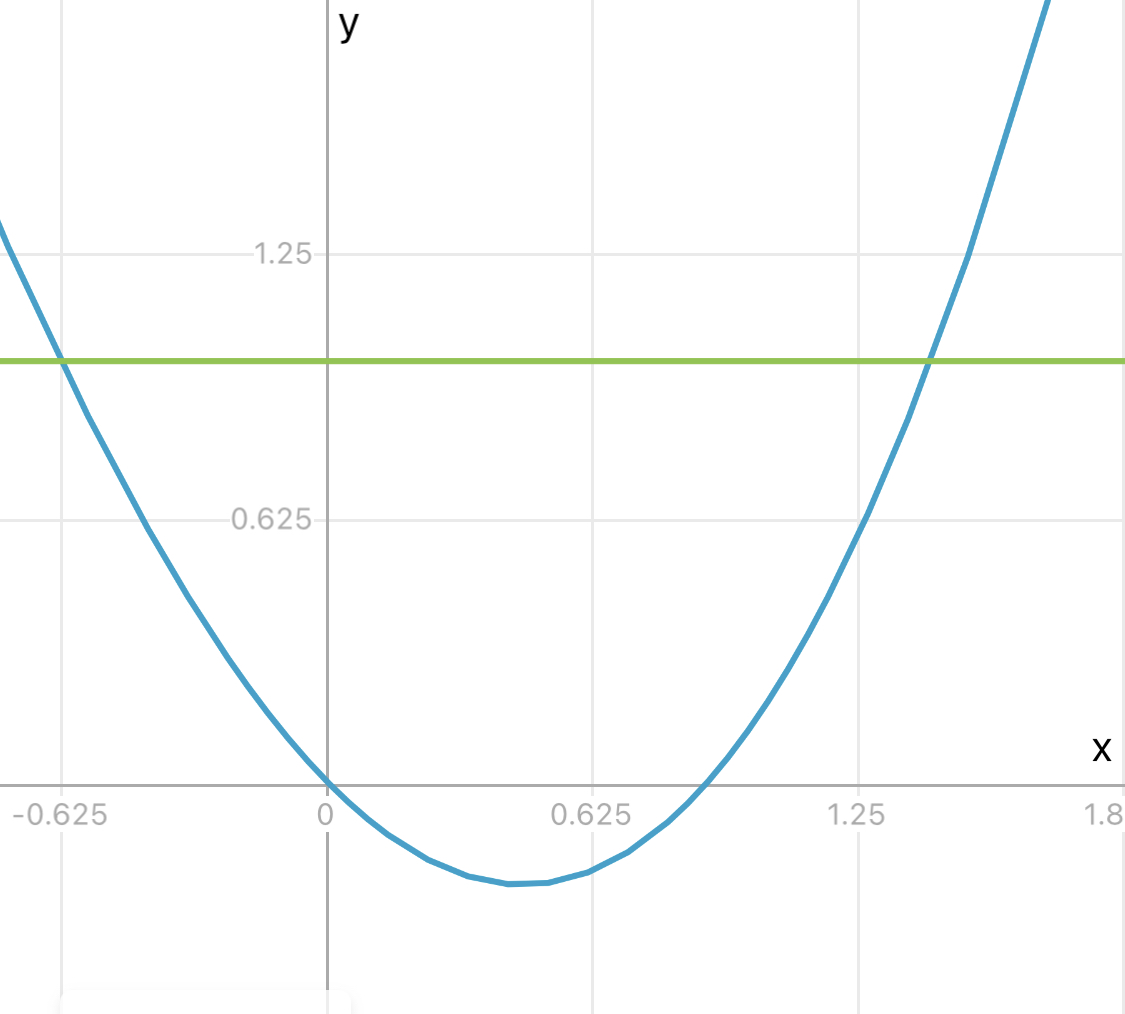
При выборe:



При выборe:



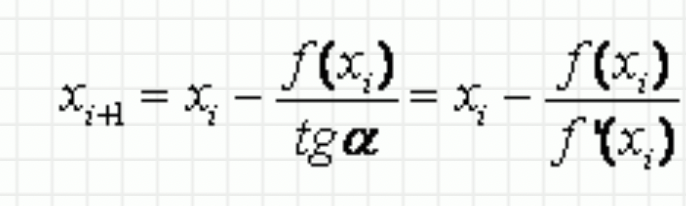
С помощью приложения photomath построим график исходного уравнения:

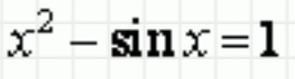


Видно, что ближе всех оказалась формула f1. Это показывает основной минус данного метода: необходимо отдельно анализировать какая итерационная формула будет точнее (и вообще подбирать формулу, которая подойдет для конкретного корня: ведь ни одна из наших формул даже при подстановке отрицательных начальных значений не дала отрицательного корня, хотя, как показывает график, он есть).

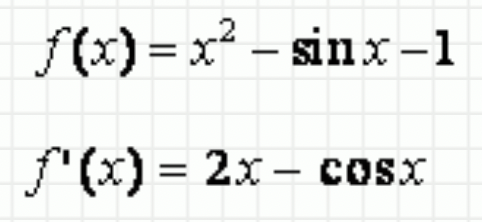
1. **Метод Ньютона (касательной)**

Алгоритм:

1. «На бумаге»: приводим уравнение к виду f(x) = 0 и вычисляем производную f ’(x)
2. Проведя математический анализ поведения функции (определить непрерывные участки, найти экстремумы и определить монотонность, и, например, по теореме Больцано-Коши сделать вывод о промежутке, на котором находится корень)
3. Вводим коэффициент погрешности, меняя который, мы будем менять точность значения x
4. Пока x отличается по модулю от своего предыдущего значения на величину, меньшую введенной нами в пункте 4, вычисляем новые значения x по формуле метода Ньютона: 

Рассмотрим работу программы на примере уравнения 

Тогда в соответствии с пунктом 1:



Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double func(double x\_pred);

double f(double x);

double f\_pr(double x);

int main(){

double x = 1; //первое приближение

double x\_pred; //переменная, куда мы будем записывать предыдущее значение x

double dx = 9999; //насколько отличается x от своего прошлого значения (изначально ставим большое число

//чтобы точно произошел вход в цикл, а далее ее значение вычисляется непосредственно)

double pogresh = 0.00001; //вводим коэффициент погрешности

int counter = 0; //счетчик количества итераций

while (fabs(dx) > pogresh){ //пока число x n-ное отстоит от x (n-1)-ное не менее, чем на dx

x\_pred = x;

x = func(x\_pred); //вычисляем следующее значение x ("следующий x") по формуле метода Ньютона

dx = x - x\_pred; //вычисляем насколько оно отстоит от предыдущего

counter++;

printf("x\_pred = %lf, x = %lf, dx = %lf\n", x\_pred, x, dx); //промежуточные значения переменных

}

printf("x = %lf (вычислено за %d итераций)", x, counter); //выводим результат

return 0;

}

double func(double x\_pred){ //формула метода Ньютона

return x\_pred - (f(x\_pred) / f\_pr(x\_pred));

}

double f(double x){ //функция возвращающая значение f(x) при подстановке туда переданного в качестве

//параметра числа

return x \* x - sin(x) - 1;

}

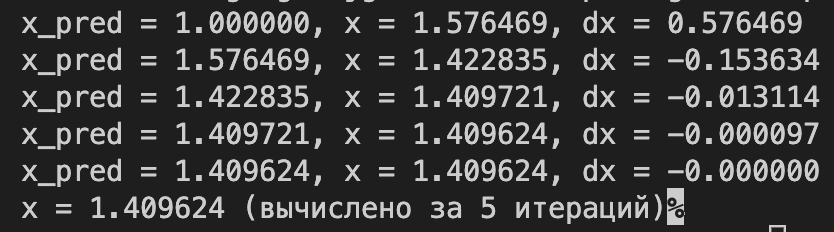
double f\_pr(double x){ //функция возвращающая значение производной f(x) при подстановке туда переданного в качестве

//параметра числа

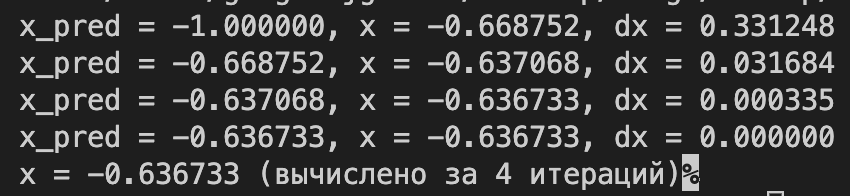
return 2 \* x - cos(x);

}

При подстановке в качестве начального приближения числа 1 получаем:



При подстановке в качестве начального приближения числа -1 получаем:

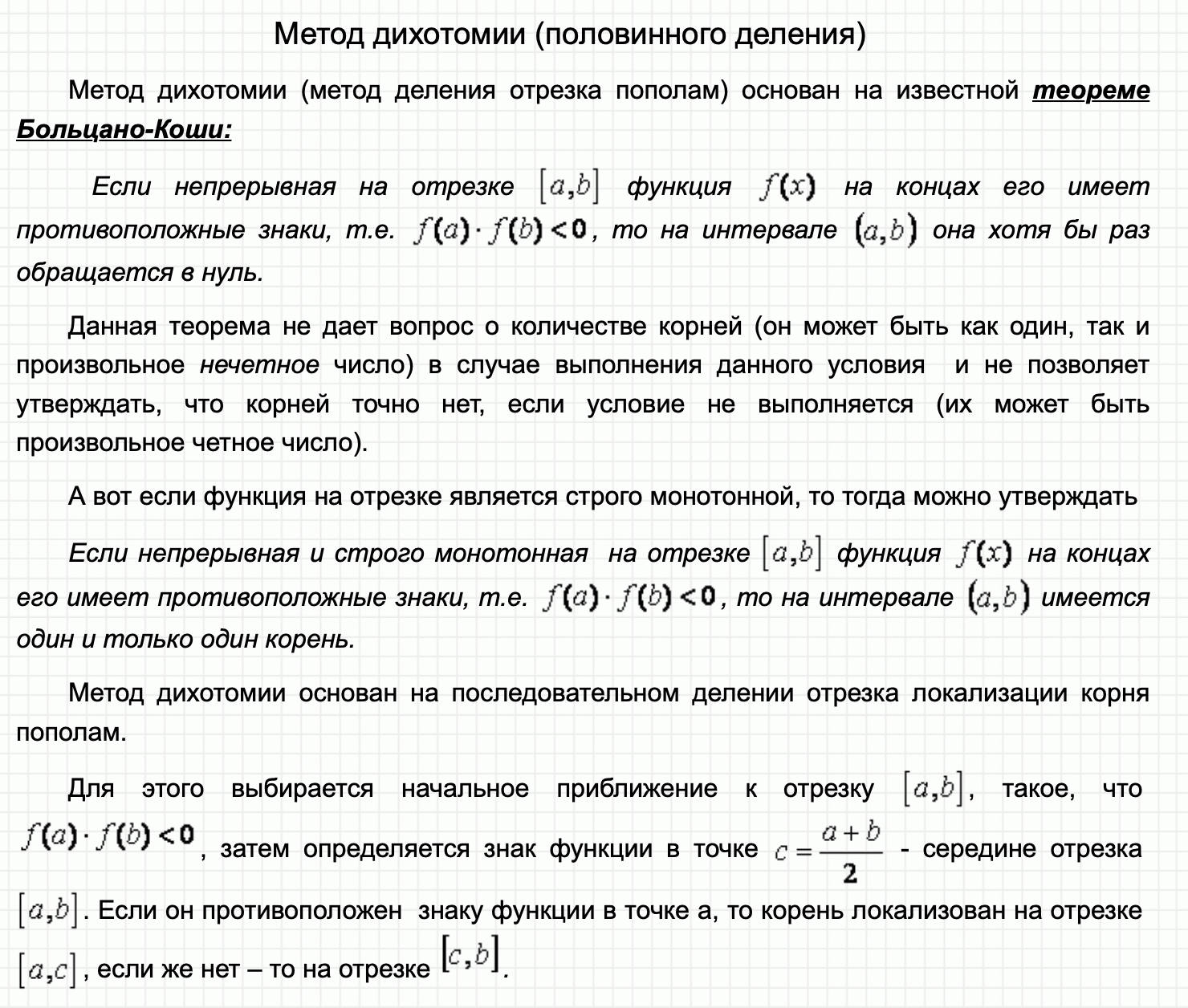


1. **Метод дихотомии**

Алгоритм:

1. Создаем переменную **a** и присваиваем ей значение аргумента, где монотонная функция принимает отрицательное значение, создаем переменную **b** и присваиваем ей значение аргумента, где монотонная функция принимает положительное значение
2. Положим с любым число не из отрезка **[ab]**
3. Пока x отличается по модулю от своего предыдущего значения на величину, меньшую величины точность, присваиваем с значение (**a** + **b**) / 2. Если и **a**, и **с** лежат слева от пересечения графика функции с осью абсцисс, то присваиваем **а** значение **с**, в противном случае, присваиваем **b** значение **c**
4. Выводи значение **с** на экран

Математическая модель:



Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double f(double x){ //возвращает значение функции от переданного аргумента

return x \* x - sin(x) - 1;

}

int main(){

double a = -1; //левая граница отрезка, где может находиться корень

double b = 0; //правая граница

double c = 10;

double c\_pred; //переменная, куда мы будем записывать предыдущее значение c

double dc = 9999; //насколько отличается c от своего прошлого значения (изначально ставим большое число

//чтобы точно произошел вход в цикл, а далее ее значение вычисляется непосредственно)

double pogresh = 0.00001; //задаем коэффициент погрешности

int counter = 0; //счетчик количества итераций

while (fabs(dc) > pogresh){ //пока число c n-ное отстоит от c (n-1)-ное не менее, чем на dc

c\_pred = c;

c = (a + b) / 2;

if (f(a) \* f(c) < 0){

b = c;

}

else{

a = c;

}

dc = c - c\_pred; //вычисляем насколько оно отстоит от предыдущего

counter++;

printf("c\_pred = %lf, c = %lf, cx = %lf\n", c\_pred, c, dc); //промежуточные значения переменных

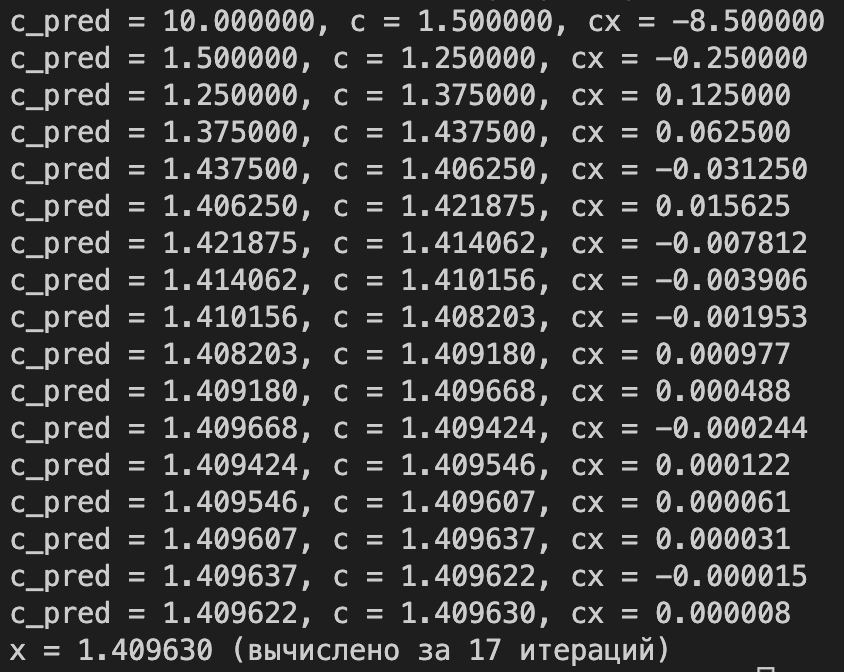
}

printf("x = %lf (вычислено за %d итераций)\n", c, counter); //выводим результат

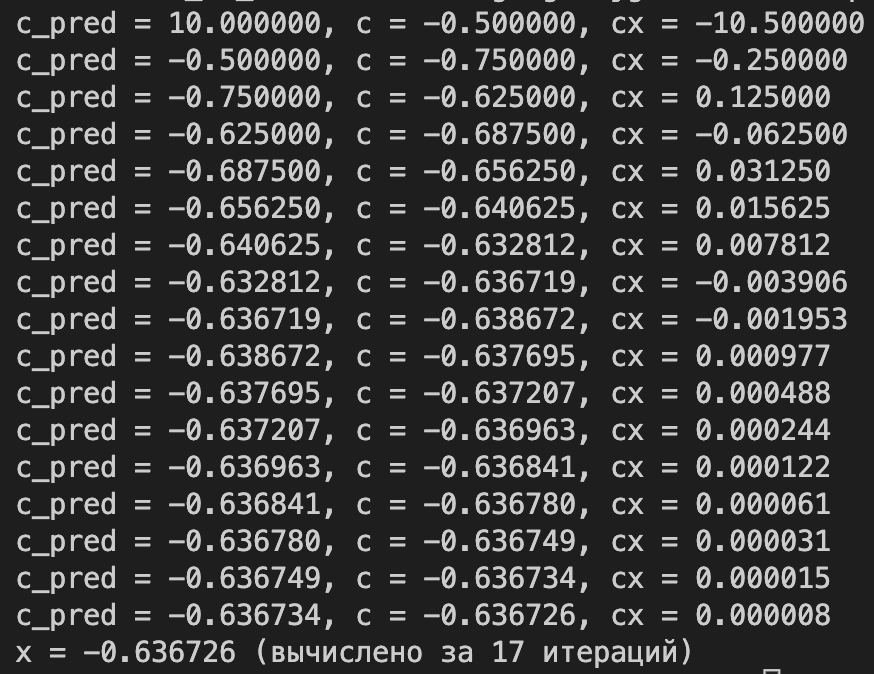
return 0;

}

При подстановке отрезка a = 0, b = 1, получаем:



При подстановке отрезка a = -1, b = 0, получаем:

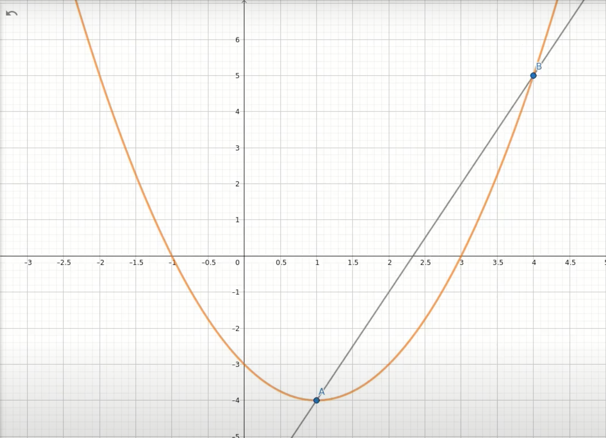
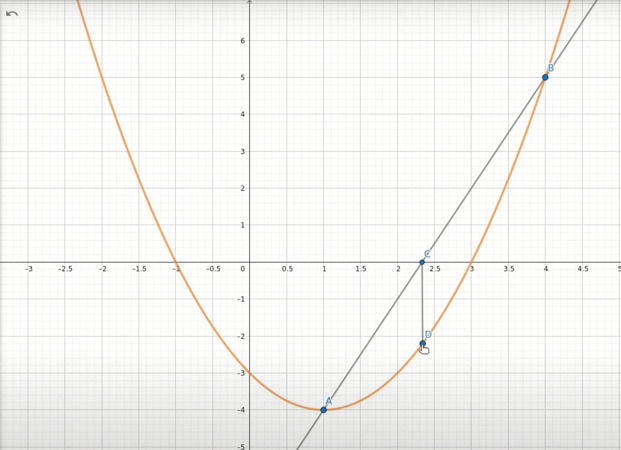


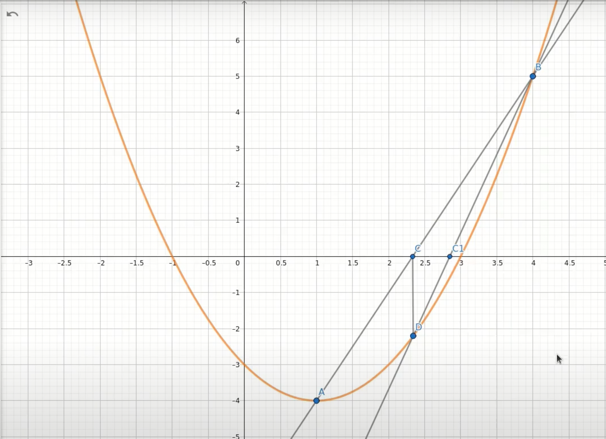
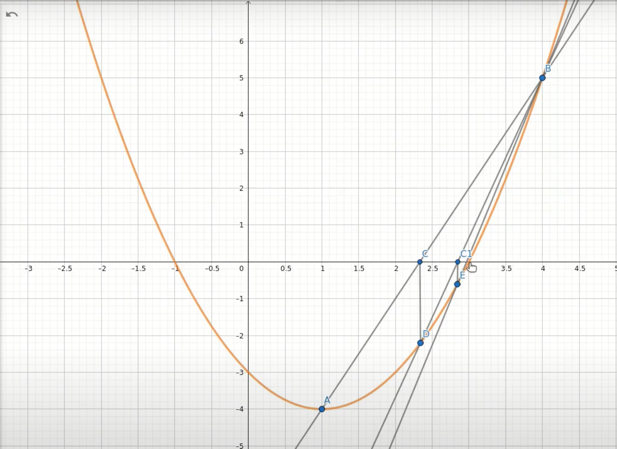
**Метод секущей**

Метод секущий по своей сути представляет из себя модифицированную версию метода дихотомии

Математическая модель:

Для начала поймем, как геометрически будет вычисляться точка c, алгоритм выглядит следующим образом:

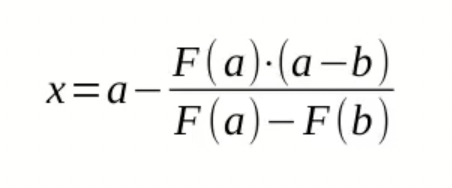
1 - Берем на графике функции две произвольные точки A и B так, чтобы значения в них имели разные знаки.

2 - Соединяем, находим точку пересечения получившейся хорды АВ с осью ОХ – точку С

3 - Находим значение функции в точке С – F(C); возвращаемся к пункту 1, взяв вместо A точку F(А), точку B не трогаем.

Происходит зацикливание, из которого мы выходим, как только добиваемся удовлетворительной точности.

Координата точки С вычисляется по формуле:



Алгоритм: см. в мат. модели и комментариях к тексту программы

Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

double f(double x){ //возвращает значение функции от переданного аргумента

return x \* x - sin(x) - 1;

}

int main(){

double a = -1; //левая граница отрезка, где может находиться корень

double b = 0; //правая граница

double c = 10; //установим произвольное значение не из отрезка AB

double c\_pred; //переменная, куда мы будем записывать предыдущее значение c

double dc = 9999; //насколько отличается c от своего прошлого значения (изначально ставим большое число

//чтобы точно произошел вход в цикл, а далее ее значение вычисляется непосредственно)

double pogresh = 0.00001; //задаем коэффициент погрешности

int counter = 0; //счетчик количества итераций

while (fabs(dc) > pogresh){ //пока число c n-ное отстоит от c (n-1)-ное не менее, чем на dc

c\_pred = c;

c = a - ((f(a) \* (a - b)) / (f(a) - f(b))); //вычисляем значение по Х точки пересечения прямой ab

//и оси ОХ

a = c;

dc = c - c\_pred; //вычисляем насколько оно отстоит от предыдущего

counter++;

printf("c\_pred = %lf, c = %lf, cx = %lf\n", c\_pred, c, dc); //промежуточные значения переменных

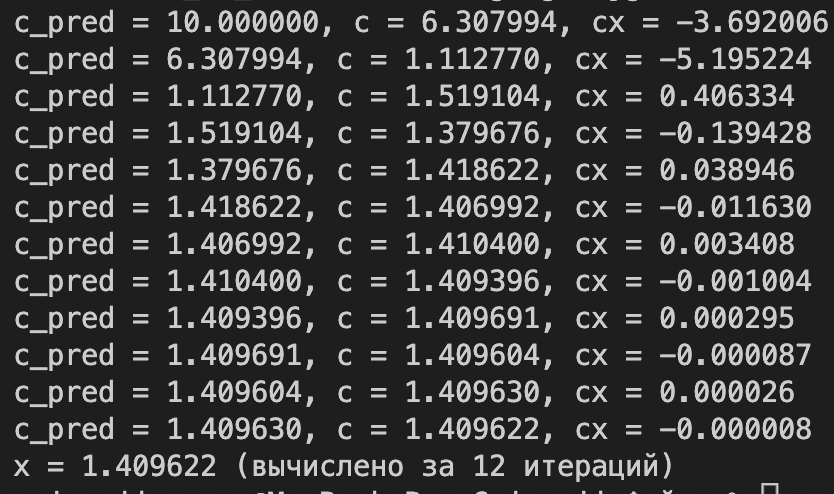
}

printf("x = %lf (вычислено за %d итераций)\n", c, counter); //выводим результат

return 0;

}

При подстановке отрезка a = 0, b = 1, получаем:



При подстановке отрезка a = -1, b = 0, получаем:

