МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №3
по дисциплине «Вычислительная математика»
Тема: Изучение методов аппроксимации значения нелинейной функции

Студент гр. 0304 Крицын Д. Р.
Преподаватель Попова Е. В.

Санкт-Петербург
2021

Цель работы. Формирование практических навыков нахождения корней алгебраических и трансцендентных уравнений методами бисекции и хорд.

Основные теоретические положения.

Метод бисекции. Если найден отрезок [a, b], такой, что $f(a)f(b) < 0$, $a_n = \xi$, $b_n = b_{n-1}$, если $f(\xi)f(a_n - 1) > 0$, Если требуется найти корень с точностью ε, то деление пополам продолжается до тех пор, пока длина отрезка не станет меньше 2ε. Тогда координата середины отрезка есть значение корня с требуемой точностью ε. Метод бисекции является простым и надежным методом поиска простого корня уравнения f(x)=0 (простым называется крень x=c дифференцируемой функции f(x), если f(c)=0 и f'(c)≠0). Этот метод сходится для любых непрерывных функций f(x), в том числе недифференцируемых. Скорость его сходимости невысока. Для достижения точности ε необходимо совершить $N \approx \log_2((b-a)/\varepsilon)$ итераций. Это означает, что для получения каждых трех верных десятичных знаков необходимо совершить около 10 итераций.

Постановка задачи. Используя программы-функции *bisect* и *_round*, найти корень уравнения методом бисекции с заданной точностью ε, исследовать зависимость числа итераций от точности ε при изменении ε от 0.1 до 0.000001, исследовать обусловленность метода (чувствительность к ошибкам в исходных данных).

1. Графически или аналитически отделить корень уравнения f(x)=0, т.е. найти отрезки [a,b], на которых функция удовлетворяет условиям теоремы Больцано-Коши ($f(a)f(b) < 0$).

2. Составить подпрограмму вычисления функции f(x).

3. Составить головную программу, содержащую обращение к подпрограмме *f*, *bisect*, *_round* и индикацию результатов.

4. Провести вычисления по программе. Построить график зависимости числа итераций от ε, сопоставить его с графиком по формуле выше.

5. Исследовать чувствительность метода к ошибкам в исходных данных. Ошибки в исходных данных моделировать с использованием подпрограммы *_round*, округляющей значения функции с заданной точностью Δ.

Метод хорд. Пусть найден отрезок [a,b], на котором функция меняет знак. Для определенности положим f(a) > 0, f(b) < 0. В методе хорд процесс итераций состоит в том, что в качестве приближений к корню уравнения f(x)=0 принимаются значения c_0, c_1, ... точек пересечения хорды с осью абсцисс. Сначала находится уравнение хорды AB: $\frac{y-f(a)}{f(b)-f(a)}=\frac{x-a}{b-a}$. Для точки пересечения ее с осью абсцисс (x = c_0, y = 0) получается уравнение: $c_0=a-\frac{b-a}{f(b)-f(a)}f(a)$. Далее сравниваются знаки величин f(a) и f(c_0) и для рассматриваемого случая оказывается, что корень находится в интервале (a, c_0), так как f(a)f(c_0) < 0. Отрезок [c_0, b] отбрасывается. Следующая итерации состоит в определении нового приближения c_1 как точки пересечения хорды AB_1 с осью абсцисс и т. д. Итерационный процесс продолжается до тех пор, пока значение f(c_n) не станет по модулю меньше заданного числа ε. Алгоритмы методов бисекции и хорд похожи, однако метод хорд в ряде случаев дает более быструю сходимость итерационного процесса, причем успех его применения, как и метода бисекции, гарантирован.

Постановка задачи. Используя функции *chord* и *_round*, найти корень уравнения f(x) = 0 заданной точностью ε методом хорд, исследовать скорость сходимости и обусловленность метода. Порядок выполнения работы:

1. Графически или аналитически отделить корень уравнения f(x)=0, т. е. найти отрезки [a,b], на которых функция f(x) удовлетворяет условиям применимости метода.

2. Составить подпрограмму-функцию вычисления функции f(x), предусмотрев округление значений функции с заданной точностью Δ с использованием программы *_round*.

3. Составить головную программу, вычисляющую корень уравнения f(x)=0 и содержащую обращение к подпрограммам *f*, *chord*, *_round* и индикацию результатов.

4. Провести вычисления по программе. Теоретически и экспериментально исследовать скорость сходимости и обусловленность метода хорд.

Выполнение работы.

$$f(x)=\frac{ctg(2x)+e\,x^2}{(-x)^2}$$

1. Локализуем корень f(x) графическим методом.

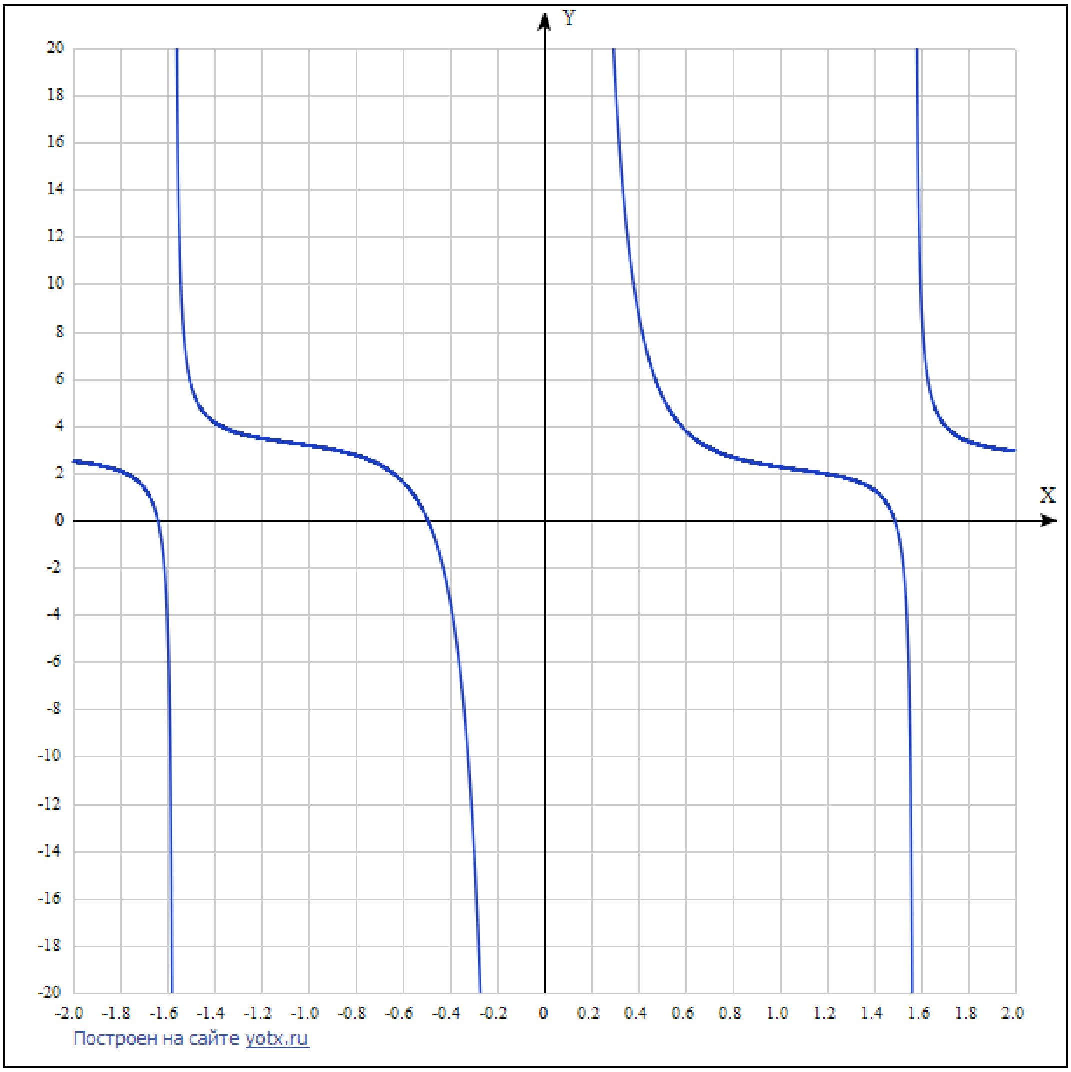

Рисунок 1 - график f(x)

$$f'(x)=(ctg(2x)+ex^2)'(-x)^{-2}+(ctg(2x)+ex^2)¿((-x)^{-2})'=$$

$$=(\frac{-2}{\sin^2(2x)}+2ex)x^{-2}+(ctg(2x)+ex^2)(-2x^{-3})=$$

$$=\frac{-\frac{2x}{\sin^2(2x)}+2ex^2-2ctg(2x)-2ex^2}{x^3}.$$

Один из корней уравнения x_0 лежит в промежутке [-1;0.3].

Будем использовать данный промежуток при дальнейших вычислениях, если не будет указано обратное.

2. Произведём аппроксимацию корня функции при помощи алгоритма бисекции, $0.000001 \le \Delta \le 0.1$, $\varepsilon = 0.001$. Реузльтаты работы программы приведены на рис. 1.

2. bisect, 0.000001 <= Δ <= 0.1, ε = 0.001, [-1;-0.3]

ε	Δ	x0	NΔ	NΔmax	Обсуловленность задачи	Кол-во итераций
0.001000	0.000010	-0.492780	0.043613	100.000000	Хорошая	9
0.001000	0.000100	-0.492900	0.043658	10.000000	Хорошая	9
0.001000	0.001000	-0.494000	0.044068	1.000000	Хорошая	9
0.001000	0.010000	-0.500000	0.046357	0.100000	Хорошая	9
0.001000	0.100000	-0.600000	0.100049	0.010000	Плохая	9

Рисунок 1 — результат работы программы для алгоритма бисекции, 0.000001 < Δ < 0.1

Видно, что меньшему значению Δ соответствует большее значение максимальной абсолютной обусловленности $\nu_{\Delta max}$, и при $\Delta \ge 0.1$ задача вычисления корня функции считается хорошо обусловленной.

3. Произведём аппроксимацию корня функции при помощи алгоритма бисекции, $0.000001 \le \varepsilon \le 0.1$, $\Delta = 0.0001$. Реузльтаты работы программы приведены на рис. 2.

3. bisect, 0.000001 <= ε <= 0.1, Δ = 0.0001, [-1;-0.3]

ε	Δ	x0	NΔ	NΔmax	Обсуловленность задачи	Кол-во итераций
0.000001	0.000100	-0.493300	0.043807	0.010000	Плохая	19
0.000010	0.000100	-0.493400	0.043844	0.100000	Хорошая	16
0.000100	0.000100	-0.493400	0.043844	1.000000	Хорошая	12
0.001000	0.000100	-0.492900	0.043658	10.000000	Хорошая	9
0.010000	0.000100	-0.486000	0.041154	100.000000	Хорошая	6
0.100000	0.000100	-0.475100	0.037427	1000.000000	Хорошая	2

Рисунок 2 — результат работы программы для алгоритма бисекции, 0.000001 < ε < 0.1

Видно, что большему значению параметра ε соответствует большее количество итераций. График зависимости количества итераций N от ε приведён на рис. 3.

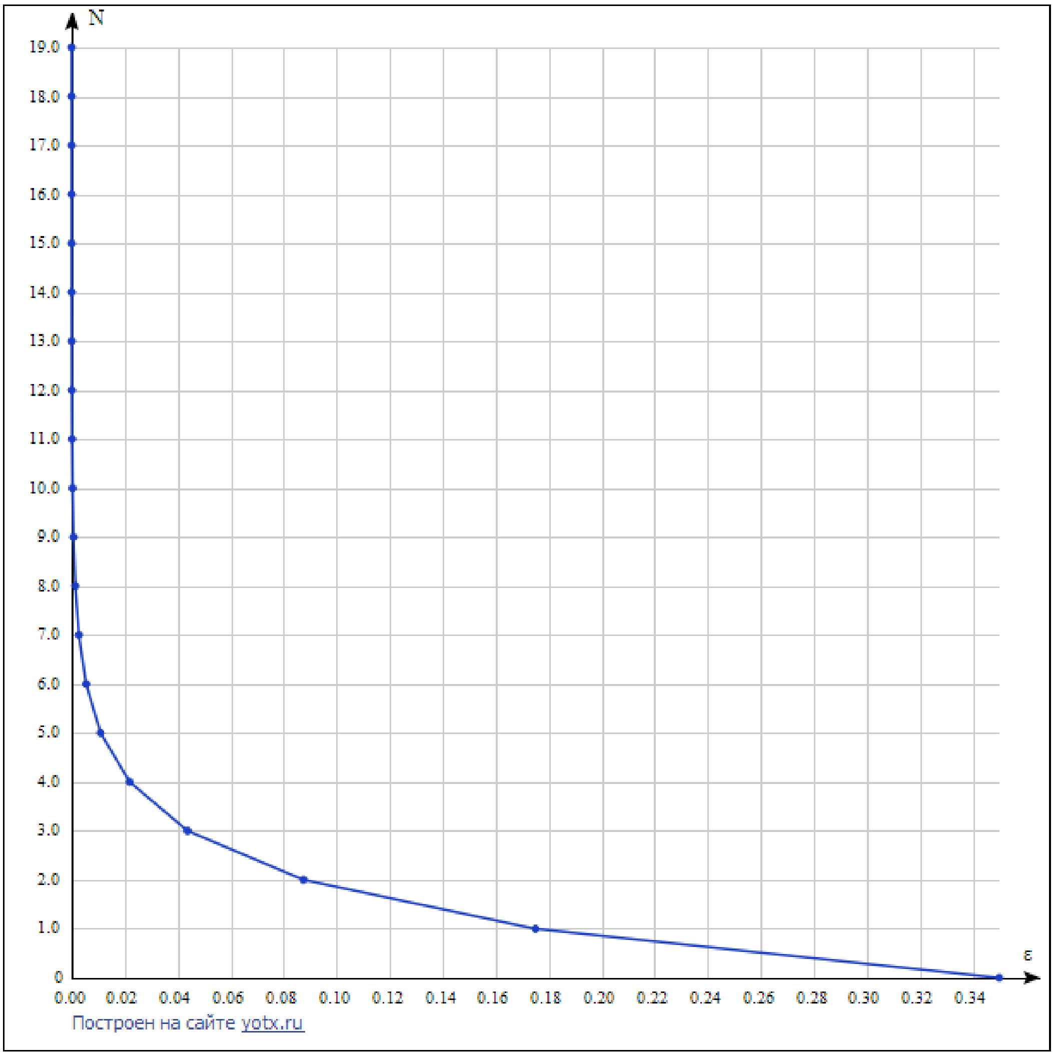

Рисунок 3 — график зависимости количества итераций N от ε для алгоритма бисекции

На данном графике видима зависимость $N \sim \log\left(\frac{1}{\varepsilon}\right)$. Это обусловлено тем, что алгоритм бисекции имеет логарифмическую сложность (каждый шаг алгоритма промежуток делится напополам), а с увеличением ε возрастает минимальный диапазон, на котором алгоритм завершает свою работу, и таким образом происходит меньше итераций.

4. Произведём аппроксимацию корня функции при помощи алгоритма бисекции, ε = 0.001, Δ = 0.0001, на промежутке [m, n], **-1.4 ≤ m ≤ -1, -0.45 ≤ n ≤ -0.3**. Реузльтаты работы программы приведены на рис. 4.

4. bisect, ε = 0.001, Δ = 0.001, [m; n], -1.4 <= m <= -1, -0.45 <= n <= -0.3

left	right	ε	Δ	x0	Кол-во итераций
-1.000000	-0.450000	0.000001	0.000100	-0.493300	19
-1.040000	-0.435000	0.000001	0.000100	-0.493300	19
-1.080000	-0.420000	0.000001	0.000100	-0.493300	19
-1.120000	-0.405000	0.000001	0.000100	-0.493300	19
-1.160000	-0.390000	0.000001	0.000100	-0.493300	19
-1.200000	-0.375000	0.000001	0.000100	-0.493300	19
-1.240000	-0.360000	0.000001	0.000100	-0.493300	19
-1.280000	-0.345000	0.000001	0.000100	-0.493300	19
-1.320000	-0.330000	0.000001	0.000100	-0.493300	19
-1.360000	-0.315000	0.000001	0.000100	-0.493300	19
-1.400000	-0.300000	0.000001	0.000100	-0.493300	20

Рисунок 4 — результат работы программы для [m, n], -1.4 ≤ m ≤ -1, -0.45 ≤ n ≤ -0.3

По полученным данным видно, что точность вычисленного значения корня функции x_0 не зависит от отрезка локализации, но его длины и границ зависит количество итераций. Чем меньше длина отрезка локализации, тем в целом быстрее работает алгоритм бисекции.

5. Произведём аппроксимацию корня функции при помощи алгоритма хорд, $0.000001 \leq \Delta \leq 0.1$, $\varepsilon = 0.001$. Реузльтаты работы программы приведены на рис. 5.

5. chord, 0.000001 <= Δ <= 0.1, ε = 0.001, [-1;-0.3]

ε	Δ	x0	NΔ	NΔmax	Обсуловленность задачи	Кол-во итераций
0.001000	0.000010	-0.493300	0.043807	100.000000	Хорошая	25
0.001000	0.000100	-0.493400	0.043844	10.000000	Хорошая	25
0.001000	0.001000	-0.494000	0.044068	1.000000	Хорошая	25
0.001000	0.010000	-0.500000	0.046357	0.100000	Хорошая	25
0.001000	0.100000	-0.600000	0.100049	0.010000	Плохая	25

Рисунок 5 — результат работы программы для алгоритма хорд, 0.000001 < Δ < 0.1

Видно, что с убыванием значения параметра Δ значение маскимальной абсолютной обусловенности $\nu_{\Delta max}$ возрастает, и при $\Delta \leq 0.01$ задача считается хорошо обусловленной.

6. Произведём аппроксимацию корня функции при помощи алгоритма хорд, $0.000001 \leq \varepsilon \leq 0.1$, $\Delta = 0.0001$. Реузльтаты работы программы приведены на рис. 6.

6. chord, 0.000001 <= ε <= 0.1, Δ = 0.001, [-1;-0.3]

ε	Δ	x0	NΔ	NΔmax	Обсуловленность задачи	Кол-во итераций
0.000001	0.000100	-0.493300	0.043807	0.010000	Плохая	43
0.000010	0.000100	-0.493300	0.043807	0.100000	Хорошая	37
0.000100	0.000100	-0.493400	0.043844	1.000000	Хорошая	31
0.001000	0.000100	-0.493400	0.043844	10.000000	Хорошая	25
0.010000	0.000100	-0.493800	0.043993	100.000000	Хорошая	19
0.100000	0.000100	-0.496400	0.044973	1000.000000	Хорошая	14

Рисунок 6 — результат работы программы для алгоритма хорд, 0.000001 < ε < 0.1

По результатам работы программы можно сделать вывод, что при возрастании параметра ε возрастает количество итераций N. График зависимости количества итераций N от ε приведён на рис. 7.

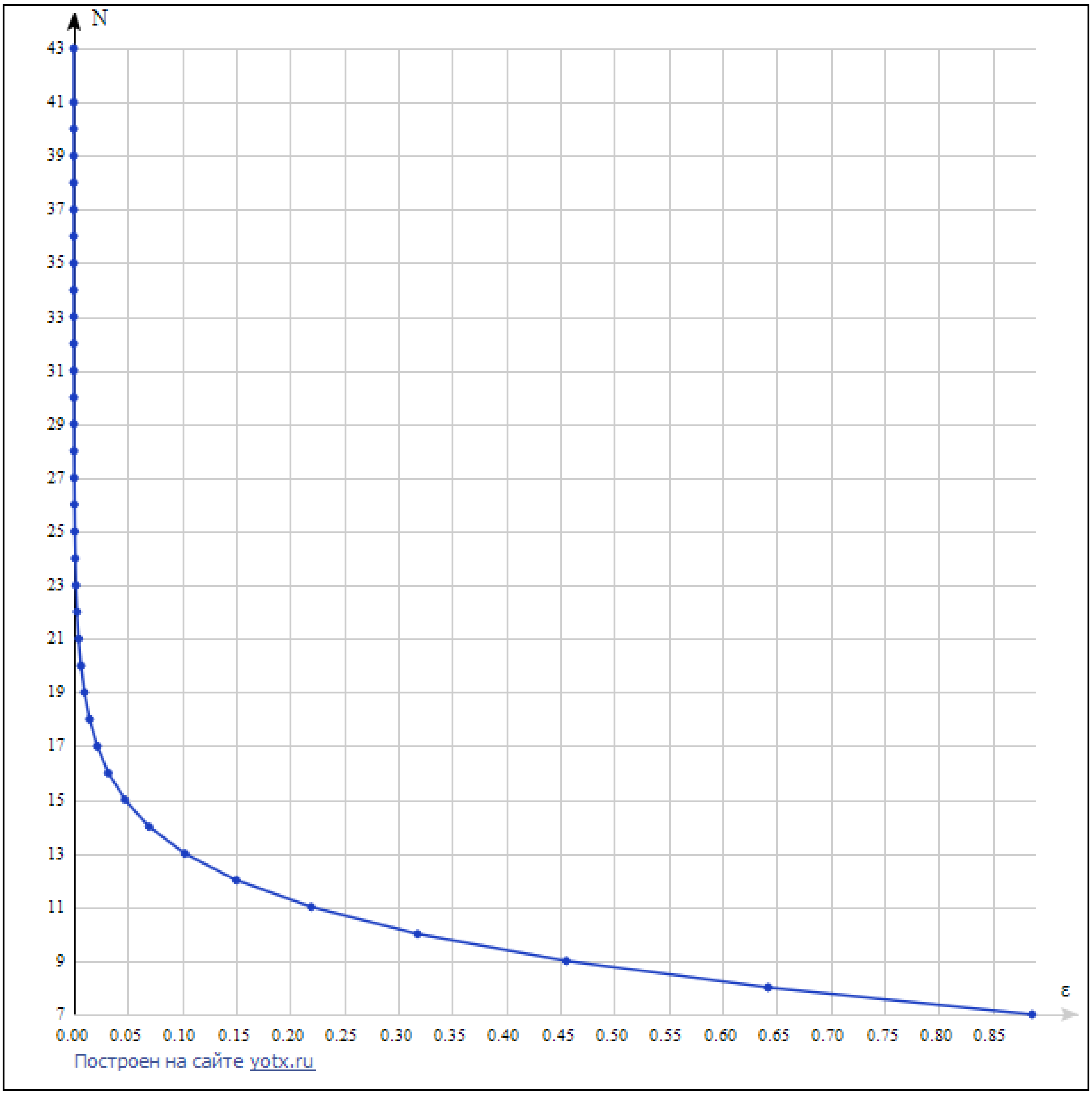

Рисунок 7 — график зависимости количества итераций N от ε для алгоритма хорд

На данном графике видима зависимость $N \sim \log\left(\frac{1}{\varepsilon}\right)$. Данная зависимость повторяет такую в алгоритме бисекции, однако в методе хорд количество итераций убывает гораздо быстрее (хотя начиная с малых значений ε оно изначально больше).

7. Произведём аппроксимацию корня функции при помощи алгоритма хорд, ε = 0.001, Δ = 0.0001, на промежутке [-1, m], -0,45 ≤ m ≤ -0,3.

Опорная, или неподвижная точка — правая, т. к. вторая производная функции f(x) < 0 в промежутке [-1, 0,3]. График второй производной приведён на рис. 8, реузльтаты работы программы приведены на рис. 9.

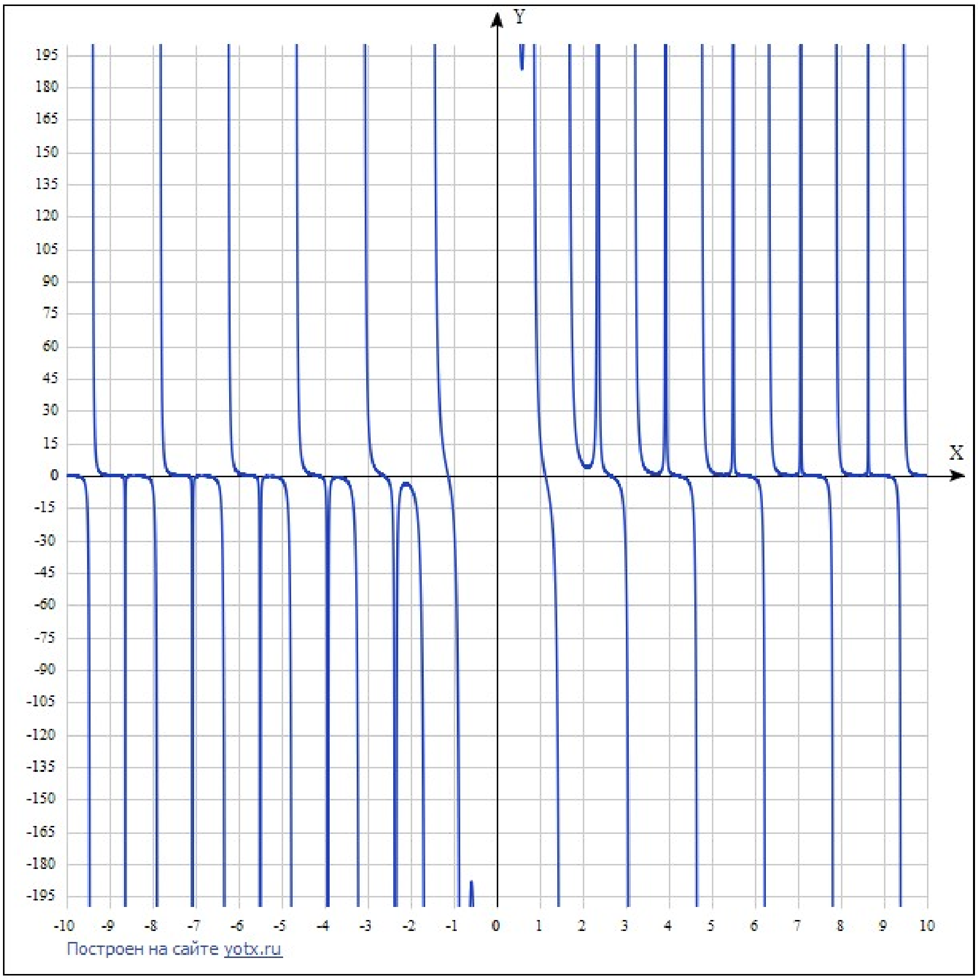

Рисунок 8 — график второй производной f(x)

7. bisect, ε = 0.001, Δ = 0.001, [-1; m], -0.45 <= m <= -0.3

left	right	ε	Δ	x0	Кол-во итераций
-1.000000	-0.450000	0.000001	0.000100	-0.493300	10
-1.000000	-0.425000	0.000001	0.000100	-0.493300	13
-1.000000	-0.400000	0.000001	0.000100	-0.493300	17
-1.000000	-0.375000	0.000001	0.000100	-0.493300	21
-1.000000	-0.350000	0.000001	0.000100	-0.493300	27
-1.000000	-0.325000	0.000001	0.000100	-0.493300	34
-1.000000	-0.300000	0.000001	0.000100	-0.493300	43

Рисунок 9 — результат работы программы для [-1, m], -0,45 ≤ m ≤ -0,3

Видно, что точность вычисленного значения зависит от точки входа. Чем ближе точка входа к корню функции, тем меньше итераций совершает алгоритм хорд.

8. Пусть Δ = 0.0001, 0.000001 ≤ ε ≤ 0.1. Результаты работы для двух аглоритмов аппроксимации представлены на рис. 10.

8. bisect, 0.000001 <= ε <= 0.01, Δ = 0.001, [-1;-0.3]

ε	Δ	bisect	x0	Кол-во итр.	chord	x0	Кол-во итр.
0.000001	0.000100		-0.493300	19		-0.493300	43
0.000010	0.000100		-0.493400	16		-0.493300	37
0.000100	0.000100		-0.493400	12		-0.493400	31
0.001000	0.000100		-0.492900	9		-0.493400	25
0.010000	0.000100		-0.486000	6		-0.493800	19
0.100000	0.000100		-0.475100	2		-0.496400	14

Рисунок 10 — результат работы программы для Δ = 0.0001, 0.000001 ≤ ε ≤ 0.1

По результатам видно, что алгоритмы бисекции и хорд дают примерно одни и те же значения корня x_0. При этом алгоритм бисекции требует большего количества итераций, чем алгоритм хорд.

Вывод.

Были изучены алгоритмы аппроксимации корня функции на примере алгоритмов бисекции и хорд и функции $f(x)=\frac{ctg(2x)+e^{x^2}}{(-x)^2}$. Было также проверено, что алгоритм бисекции сходится куда медленнее, чем алгоритм хорд, и это действительно верно для достаточно больших значений параметра ε, при этом точность и обусловленность данных алгоритмов примерно одинаковы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <math.h>
#include <errno.h>

#define E ((double)2.71828)

#define ERRNO_APPROX_INTERVAL 1
#define ERRNO_APPROX_PRECISION 2

#define ERRNO_ROUND_PRECISION 2

double _round(double x, double delta)
{
  if(delta <= 1e-9){
  errno = ERRNO_ROUND_PRECISION;
  return NAN;
  }
  return delta * floor((x / delta) + (x > 0 ? 0.5 : -0.5));
}


double bisect(double (*f)(double),
  double left, double right,
  double eps, int* N)
{
  double e = fabs(eps) * 2;
  double fleft = f(left);
  double fright = f(right);
  double x = (left + right) / 2, y;

  if(fleft*fright > 0){
  errno = ERRNO_APPROX_INTERVAL;
  return NAN;
  }
  if(eps <= 0){
  errno = ERRNO_APPROX_PRECISION;
  return NAN;
  }

  *N = 0;

  if(fleft == 0) return left;
  if(fright == 0) return right;

  while((right - left) >= e)
  {
  x = (left + right) / 2;
  y = f(x);
  if(y == 0)
  return x;
  if(y*fleft < 0)
  right = x;
  else {
  left = x;
  fleft = y;
  }

  (*N)++;
```

```
  }

  return x;
}

double chord(double (*f)(double), double left, double right, double eps, int* N)
{
  double fleft = f(left);
  double fright = f(right);
  double x, y;

  if(fleft*fright > 0){
  errno = ERRNO_APPROX_INTERVAL;
  return NAN;
  }
  if(eps <= 0){
  errno = ERRNO_APPROX_PRECISION;
  return NAN;
  }

  *N = 0;

  if(fleft == 0) return left;
  if(fright == 0) return right;

  do{
  x = left - (right - left) * fleft / (fright - fleft); // предполагаемый корень
хорды
  y = f(x);

  if(y == 0)
  return x;
  if(y * fleft < 0){
  right = x;
  fright = y;
  }
  else{
  left = x;
  fleft = y;
  }

  (*N)++;
  } while(fabs(y) >= eps);

  return x;
}


double f(double x)
{
  return (1/tan(2*x) + E * pow(x, 2)) / pow(-x, 2);
}
double f_conditioning(double x)
{
  return 1 / fabs(-2*(x*pow(1/sin(2*x), 2) + 1/tan(2*x)) / pow(x, 3));
}


int main()
{
  double left = -1, right = -0.3;

  printf("2. bisect, 0.000001 <= Δ <= 0.1, ε = 0.001, [%lg;%lg]\n", left, right);
```

```
	puts("ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|Обсуловленность задачи\t\t|Кол-во итераций");
	for(double eps = 0.001, delta = 0.00001; delta <= 0.1; delta *= 10)
	{
	int N;
	double x0 = bisect(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", eps, delta, x0, cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	printf("\n3. bisect, 0.000001 <= ε <= 0.1, Δ = 0.0001, [%lg;%lg]\n", left, right);
	puts("ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|Обсуловленность задачи\t\t|Кол-во итераций");
	for(double eps = 0.000001, delta = 0.0001; eps <= 0.1; eps *= 10)
	{
	int N;
	double x0 = bisect(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", eps, delta, x0, cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	puts("\n3. График зависимости N от ε");
	int prev_N = -1;
	for(double eps = 0.000001, delta = 0.0001; eps <= 1.0; eps += 0.000001)
	{
	int N;
	double x0 = bisect(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	if(N != prev_N)
	printf("(%lf;%d)\n", eps, N);
	prev_N = N;
	if(N == 0)
	break;
	}
	puts("\n4. bisect, ε = 0.001, Δ = 0.001, [m; n], -1.4 <= m <= -1, -0.45 <= n <= -0.3");
	puts("|left\t\t|right\t\t|ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|Обсуловленность задачи\t\t|Кол-во итераций");
	for(double eps = 0.000001, delta = 0.0001, left = -1, right = -0.45; left >= -1.41 && right <= -0.29; left += -(1.4 - 1) / 10, right += (0.45 - 0.3) / 10)
	{
	int N;
	double x0 = bisect(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("|%lf\t|%lf\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", left, right, eps, delta, x0, cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	left = -1;
	right = -0.3;
	printf("5. chord, 0.000001 <= Δ <= 0.1, ε = 0.001, [%lg;%lg]\n", left, right);
	puts("ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|Обсуловленность задачи\t\t|Кол-во итераций");
```

```
	for(double eps = 0.001, delta = 0.00001; delta <= 0.1; delta *= 10)
	{
	int N;
	double x0 = chord(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", eps, delta, x0,
cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	printf("\n6. chord, 0.000001 <= ε <= 0.1, Δ = 0.001, [%lg;%lg]\n", left, right);
	puts("ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|Обсуловленность задачи\t\t|Кол-
во итераций");
	for(double eps = 0.000001, delta = 0.0001; eps <= 0.1; eps *= 10)
	{
	int N;
	double x0 = chord(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", eps, delta, x0,
cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	puts("\n6. График зависимости N от ε");
	prev_N = -1;
	for(double eps = 0.000001, delta = 0.0001; eps <= 1.0; eps += 0.000001)
	{
	int N;
	double x0 = chord(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	if(N != prev_N)
	printf("(%lf;%d)\n", eps, N);
	prev_N = N;
	if(N == 0)
	break;
	}
	printf("\n7. bisect, ε = 0.001, Δ = 0.001, [%lg; -0.3], -1.4 <= m <= -1\n", left);
	puts("|left\t\t|right\t\t|ε\t\t\t|Δ\t\t\t|x0\t\t\t|NΔ\t\t\t|NΔmax\t\t\t|
Обсуловленность задачи\t\t|Кол-во итераций");
	for(double eps = 0.000001, delta = 0.0001, left = -1.4; left <= -1; left += 0.05)
	{
	int N;
	double x0 = chord(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
	double cond_max = eps/delta;

	printf("|%lf\t|%lf\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%lf\t\t|%s\t\t\t|%d\n", left,
right, eps, delta, x0, cond, cond_max, cond <= cond_max ? "Хорошая" : "Плохая ", N);
	}
	left = -1;
	printf("\n8. bisect, 0.000001 <= ε <= 0.01, Δ = 0.001, [%lg;%lg]\n", left, right);
	puts("ε\t\t|Δ\t\t||bisect\t|x0\t\t|Кол-во итр.\t||chord\t|x0\t\t|Кол-во итр.\t");
	for(double eps = 0.000001, delta = 0.0001; eps <= 0.1; eps *= 10)
	{
	int N;
	double x0 = bisect(&f, left, right, eps, &N);
	x0 = _round(x0, delta);
	double cond = f_conditioning(x0);
```

```
  double cond_max = eps/delta;

  printf("%lf\t|%lf\t\t\t|%lf\t|%d", eps, delta, x0, N);

  x0 = chord(&f, left, right, eps, &N);
  x0 = _round(x0, delta);

  printf("\t\t\t|%lf\t|%d\n", x0, N);
  }
}
```