Московский Авиационный Институт

(национальный исследовательский университет)

Факультет «Прикладной математики и информатики»

Кафедра 806

Курсовой проект по теме: «Процедуры и функции в качестве параметров»

Выполнил: студент 1 курса

Каширин К.Д.

Преподаватель: Трубченко Н.М.

Москва 2020

Содержание

Цель

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления — дихотомии). Нелинейные уравнения оформить как параметры-функции, разрешив относительно неизвестной величины в случае необходимости. Применить каждую процедуру к решению двух уравнений, заданных двумя строками таблицы, начиная с варианта с заданным номером.

Метод Итераций

Идея метода заключается в замене исходного уравнения $F(x) = 0$ уравнением вида $x = f(x)$.

Достаточное условие сходимости метода:$|f'(x)| < 1$, $x \in [a, b]$. Это условие необходимо проверить перед началом решения задачи, так как функция $f(x)$ может быть выбрана неоднозначно, причем в случае неверного выбора указанной функции метод расходится.

Начальное приближение корня: $x^{(0)} = (a + b)/2$ (середина исходного отрезка).

Итерационный процесс: $x^{(k+1)} = f(x^{(k)})$

Условие окончания: $\left|x^{(k)} - x^{(k+1)}\right| < \varepsilon$

Приближенное значение корня:$x^* \approx x^{(\text{конечное})}$

Вариант 12.

12	$\ln x - x + 1{,}8 = 0$	[2, 3]	итераций	2.8459

На языке Си метод итераций реализован:

```
double it(double x, double y,double eps){
    for (unsigned i = 1;fabs(x-y)>=eps*100;i+=1){
        x = y;
        y = f(x);
        printf("%d | %lf %lf %lf\n",i,y,2.8459,y-2.8459);
    }
    return y;
}
```

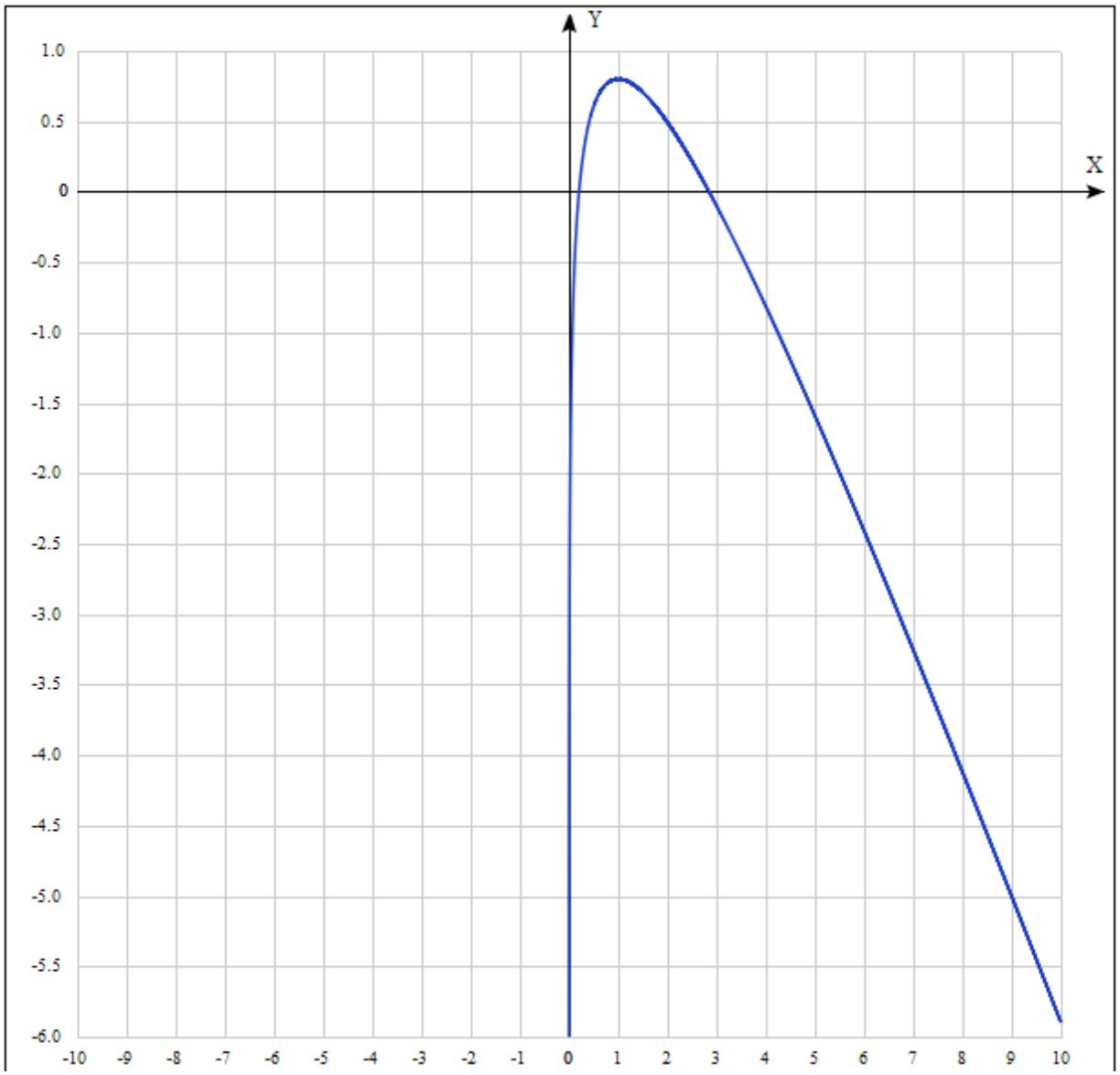
Функция возвращает корень уравнения (вещественное число). На вход подаётся начальное приближение корня, значение функции в точке начального приближения корня и машинное эпсилон. При этом, функцией является выраженный корень одного из слагаемых в данном уравнении(т.е. $f(x) = \ln x + 1{,}8$)

Создаем цикл, который будет работать до тех пор, пока значения функция между двумя корнями будут больше чем $\varepsilon * k$ (k = 100). В цикле выводится: номер шага, x^k, приближенное значение корня, разница между приближенным значением корня и x^k.

Протокол программы:

```
kirill@LAPTOP-F153AKTP:~$ gcc kp4_var12.c -lm
kirill@LAPTOP-F153AKTP:~$ ./a.out
1 | 2.799267 2.845900 -0.046633
2 | 2.829358 2.845900 -0.016542
3 | 2.840050 2.845900 -0.005850
4 | 2.843822 2.845900 -0.002078
5 | 2.845149 2.845900 -0.000751
6 | 2.845615 2.845900 -0.000285
7 | 2.845779 2.845900 -0.000121
8 | 2.845837 2.845900 -0.000063
9 | 2.845857 2.845900 -0.000043
10 | 2.845864 2.845900 -0.000036
11 | 2.845867 2.845900 -0.000033
12 | 2.845868 2.845900 -0.000032
13 | 2.845868 2.845900 -0.000032
14 | 2.845868 2.845900 -0.000032
15 | 2.845868 2.845900 -0.000032
16 | 2.845868 2.845900 -0.000032
17 | 2.845868 2.845900 -0.000032
18 | 2.845868 2.845900 -0.000032
19 | 2.845868 2.845900 -0.000032
20 | 2.845868 2.845900 -0.000032
21 | 2.845868 2.845900 -0.000032
22 | 2.845868 2.845900 -0.000032
23 | 2.845868 2.845900 -0.000032
24 | 2.845868 2.845900 -0.000032
25 | 2.845868 2.845900 -0.000032
26 | 2.845868 2.845900 -0.000032
27 | 2.845868 2.845900 -0.000032
28 | 2.845868 2.845900 -0.000032
29 | 2.845868 2.845900 -0.000032
x = 2.845868
```

График функции:



Полный текст программы:

```c
#include <stdio.h>
#include <math.h>
double f(double x){
    return log(x)+1.8;
}
double derivative(double x){
    return 1/x-1;
}
double macheps(){
    double e = 1.0;
    while (1.0 + e / 2.0 > 1.0)
        e /= 2.0;
    return e;
}
double it(double x, double y,double eps){
    for (unsigned i = 1;fabs(x-y)>=eps*100;i+=1){
        x = y;
        y = f(x);
        printf("%d | %lf %lf %lf\n",i,y,2.8459,y-2.8459);
    }
    return y;
}
int main(){
    double a = 2;
    double b = 3;
    double x = (a+b)/2;
    double y = f(x);
    double eps = macheps();
    double answer = it(x,y,eps);
    printf("x = %lf\n",answer);
}
```

Метод дихотомии

Очевидно, что если на отрезке $[a, b]$ существует корень уравнения, то значения функции на концах отрезка имеют разные знаки:$F(a) * F(b) < 0$. Метод заключается в делении отрезка пополам и его сужении в два раза на каждом шаге итерационного процесса в зависимости от знака функции в середине отрезка.

Итерационный процесс строится следующим образом: за начальное приближение принимаются границы исходного отрезка $a^{(0)} = a, b^{(0)} = b$. Далее вычисления проводятся по формулам:$a^{(k+1)} = (a^k + b^{(k)})/2$, $b^{(k+1)} = b^{(k)}$, если $F(a^{(k)})*F\left(\frac{a^{(k)}+b^{(k)}}{2}\right) >0$; или по формулам:$a^{(k+1)} = a^{(k)}, b^{(k+1)} = (a^{(k)} + b^{(k)})/2$, если $F\left(b^{(k)}\right) * F\left(\frac{a^{(k)}+b^{(k)}}{2}\right) > 0$

Процесс повторяется до тех пор, пока не будет выполнено условие окончания $\left|a^{(k)} - b^{(k)}\right| < \varepsilon$

Приближенное значение корня к моменту окончания итерационного процесса получается следующим образом $x^* \approx \left(a^{(\text{конечное})} + b^{(\text{конечное})}\right)/2$.

Вариант 11.

11	$e^x + \sqrt{1 + e^{2x}} - 2 = 0$	[-1, 0]	дихотомии	-0.2877

На языке Си метод реализован:

```
double d(double x,double y,double eps,int n,double k){
    for (double i = 0 ;fabs(x-y)>=eps*100;){
        if ((f(x)*f((x+y)/2))>0){
            x=(x+y)/2;k=x;
        } else if ((f(y)*f((x+y)/2))>0) {
            y=(x+y)/2;k=y;
        }
        n+=1;
        printf("%d | %lf %lf %lf\n",n,k,-0.2877,k+0.2877);
    }
    return x;
}
```

Функция возвращает корень уравнения (вещественное число). На вход подается левая граница отрезка, правая граница отрезка, машинное эпсилон, количество итераций, переменная k, которая будет хранить значение x или y (в зависимости от выполнения условия) на каждом шаге. Цикл работает до тех пор пока разница между границами отрезка не будет меньше, чем $\varepsilon * \mu$(в нашем случае $\mu = 100$). В цикле выводится: номер шага, k, разница между k и приближенном значением корня. Каждый раз мы одной из границ,

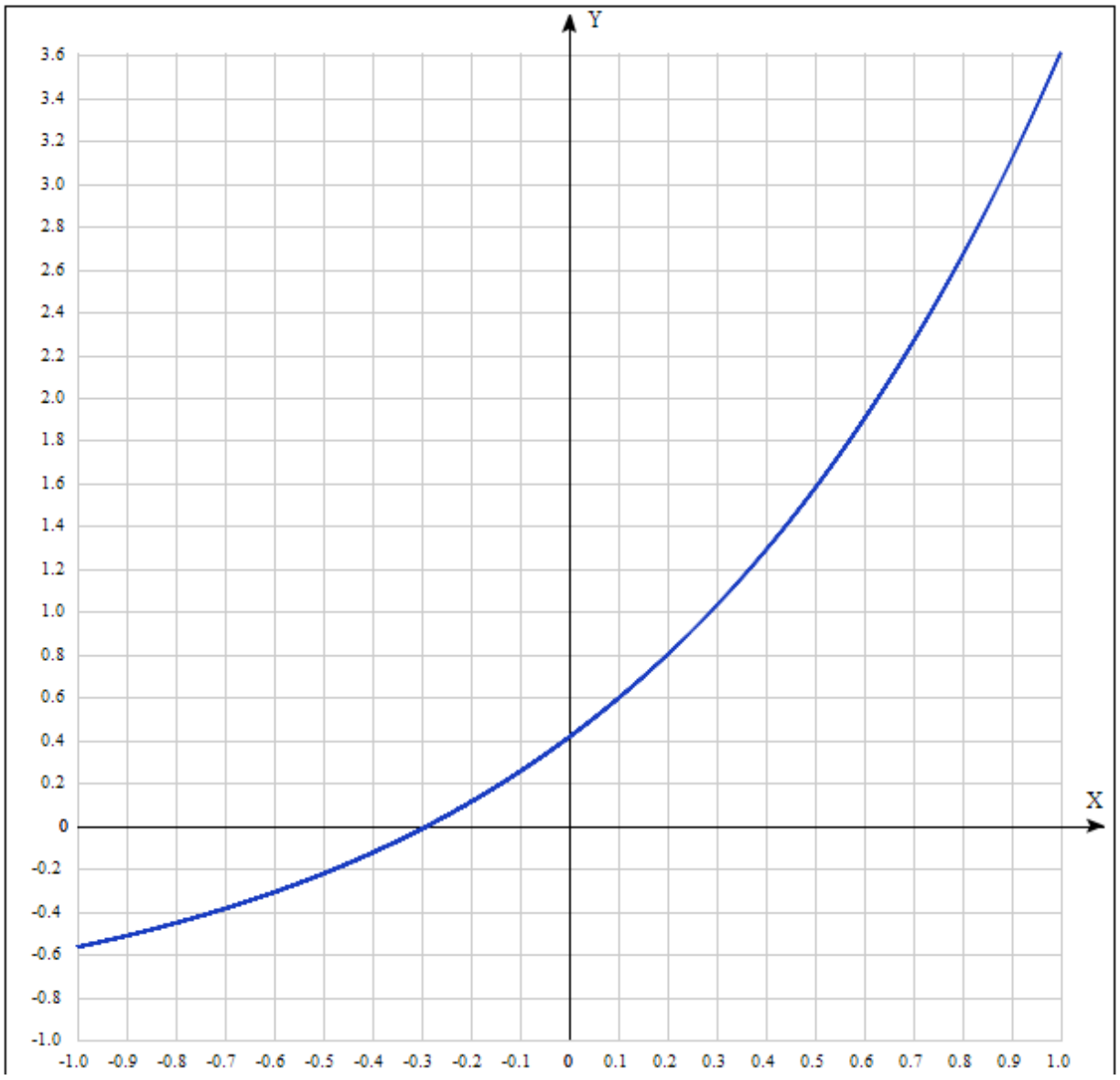
в зависимости от условия, присваиваем значение среднего арифметического этих границ.

Протокол программы:

```
kirill@LAPTOP-F153AKTP:~$ gcc kp4_var11.c -lm
kirill@LAPTOP-F153AKTP:~$ ./a.out
1 | -0.500000 -0.287700 -0.212300
2 | -0.250000 -0.287700 0.037700
3 | -0.375000 -0.287700 -0.087300
4 | -0.312500 -0.287700 -0.024800
5 | -0.281250 -0.287700 0.006450
6 | -0.296875 -0.287700 -0.009175
7 | -0.289062 -0.287700 -0.001362
8 | -0.285156 -0.287700 0.002544
9 | -0.287109 -0.287700 0.000591
10 | -0.288086 -0.287700 -0.000386
11 | -0.287598 -0.287700 0.000102
12 | -0.287842 -0.287700 -0.000142
13 | -0.287720 -0.287700 -0.000020
14 | -0.287659 -0.287700 0.000041
15 | -0.287689 -0.287700 0.000011
16 | -0.287674 -0.287700 0.000026
17 | -0.287682 -0.287700 0.000018
18 | -0.287685 -0.287700 0.000015
19 | -0.287683 -0.287700 0.000017
20 | -0.287683 -0.287700 0.000017
21 | -0.287682 -0.287700 0.000018
22 | -0.287682 -0.287700 0.000018
23 | -0.287682 -0.287700 0.000018
24 | -0.287682 -0.287700 0.000018
25 | -0.287682 -0.287700 0.000018
26 | -0.287682 -0.287700 0.000018
27 | -0.287682 -0.287700 0.000018
28 | -0.287682 -0.287700 0.000018
29 | -0.287682 -0.287700 0.000018
30 | -0.287682 -0.287700 0.000018
31 | -0.287682 -0.287700 0.000018
32 | -0.287682 -0.287700 0.000018
33 | -0.287682 -0.287700 0.000018
34 | -0.287682 -0.287700 0.000018
35 | -0.287682 -0.287700 0.000018
36 | -0.287682 -0.287700 0.000018
37 | -0.287682 -0.287700 0.000018
38 | -0.287682 -0.287700 0.000018
39 | -0.287682 -0.287700 0.000018
40 | -0.287682 -0.287700 0.000018
41 | -0.287682 -0.287700 0.000018
42 | -0.287682 -0.287700 0.000018
43 | -0.287682 -0.287700 0.000018
```

```
44 | -0.287682 -0.287700 0.000018
45 | -0.287682 -0.287700 0.000018
46 | -0.287682 -0.287700 0.000018
x = -0.287682
```

График функции:



Программа:

```c
#include <stdio.h>
#include <math.h>
double f(double i){
    return exp(i)+sqrt(1+exp(2*i))-2;
}
double macheps(){
    double e = 1.0;
    while (1.0 + e / 2.0 > 1.0)
        e /= 2.0;
    return e;
}
double d(double x,double y,double eps,int n,double k){
    for (double i = 0 ;fabs(x-y)>=eps*100;){
        if ((f(x)*f((x+y)/2))>0){
            x=(x+y)/2;k=x;
        } else if ((f(y)*f((x+y)/2))>0) {
            y=(x+y)/2;k=y;
        }
        n+=1;
        printf("%d | %lf %lf %lf\n",n,k,-0.2877,k+0.2877);
    }
    return x;
}
int main(){
    int n=0;
    double a = -1;
    double b = 0;
    double x,y,k;
    double eps=macheps();
    x=a;y=b;
    double answer = d(x,y,eps,n,k);
    printf("x = %lf\n",answer);
}
```

Метод Ньютона

Метод Ньютона является частным случаем метода итераций.

Условие сходимости метода: $|F(x) * F''(x)| < (F'(x))^2$ на отрезке [a, b].

Итерационный процесс: $x^{(k+1)} = x^{(k)} - F(x^{(k)})/F'(x^{(k)})$

Вариант 10.

10	$2x \cdot \sin x - \cos x = 0$	[0.4, 1]	Ньютона	0.6533

На языке Си метод реализован:

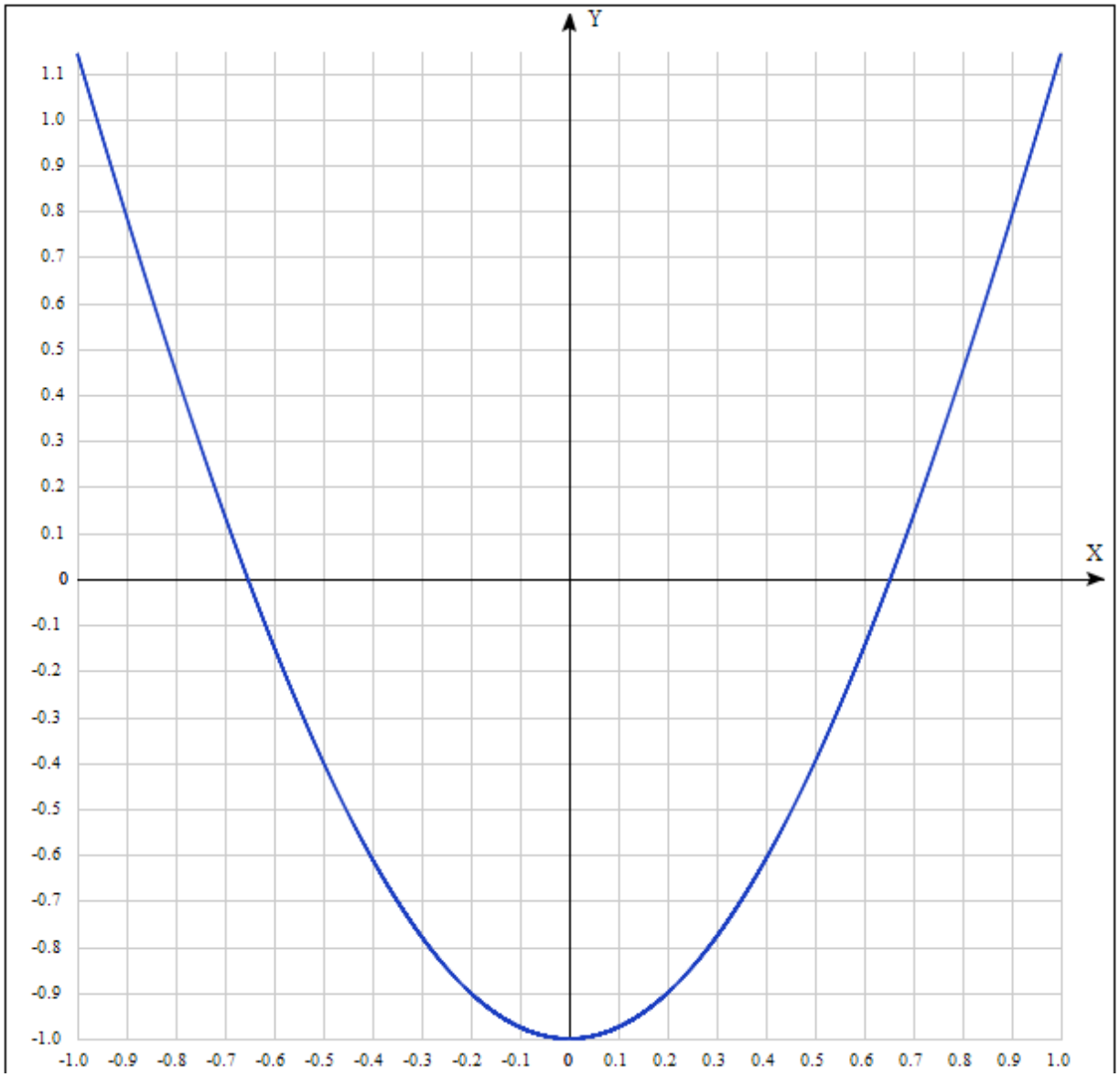
```
double newton(double x, double y , double eps){
    for (unsigned i = 2;fabs(f(x)*d1(x))>=d(x)*d(x) || (fabs(x-y)>=eps*100);i+=1){
        y = x;
        x = y - f(x)/d(x);
        printf("%d | %lf %lf %lf\n",i,x,0.6533,x-0.6533);
    }
    return x;
}
```

Функция возвращает корень уравнения (вещественное число). На вход подается среднее арифметическое левой и правой границы, переменная y, которая изначально определена как среднее арифметическое левой и правой границы, а впоследствии будет определять предыдущее значение x, машинное эпсилон. В цикле переменная y приобретает значение x, а переменная x уменьшается на величину, равную отношение производной первого и второго порядка. В цикле выводится: номер шага, значение x, приближенное значение корня, разница между x и приближенным значением корня. Цикл работает до тех пор, пока разница между предыдущим значением корня и текущим будет не больше, чем $\varepsilon * k$(в нашем случае $k = 100$).

Протокол программы:

```
kirill@LAPTOP-F153AKTP:~$ gcc kp4_var10.c -lm
kirill@LAPTOP-F153AKTP:~$ ./a.out
1 | 0.700000 0.653300 0.046700
2 | 0.654365 0.653300 0.001065
3 | 0.653272 0.653300 -0.000028
4 | 0.653271 0.653300 -0.000029
5 | 0.653271 0.653300 -0.000029
6 | 0.653271 0.653300 -0.000029
x = 0.653271
```

График функции:



Программа:

```c
#include <stdio.h>
#include <math.h>
double f(double x){
    return 2*x*sin(x)-cos(x);
}
double d(double x){
    return 3*sin(x)+2*x*cos(x);
}
double d1(double x){
    return 5*cos(x)-2*x*sin(x);
}
double macheps(){
    double e = 1.0;
    while (1.0 + e / 2.0 > 1.0)
        e /= 2.0;
    return e;
}
double newton(double x, double y , double eps){
    for (unsigned i = 2;fabs(f(x)*d1(x))>=d(x)*d(x) || (fabs(x-y)>=eps*100);i+=1){
        y = x;
        x = y - f(x)/d(x);
        printf("%d | %lf %lf %lf\n",i,x,0.6533,x-0.6533);
    }
    return x;
}
int main(){
    double a = 0.4;
    double b = 1;
    double x = (a+b)/2;
    double y;
    double eps = macheps();
    printf("%d | %lf %lf %lf\n",1,x,0.6533,x-0.6533);
    double answer=newton(x,y,eps);
    printf("x = %lf\n",answer);
}
```

Вывод

Составил программу на Си, вычислил корни заданных уравнений различными методами (методом дихотомии, методом Ньютона, методом итераций), есть небольшая погрешность между найденным значением корня и приближенным значением корня, который был дан в условии.