**Московский Авиационный Институт**

**(Национальный исследовательский университет)**

**Институт № 8 «Информационные технологии и прикладная математика»**

**Кафедра №806 «Вычислительная математика и программирование»**

# КУРСОВОЙ ПРОЕКТ

по курсам  
«Фундаментальная информатика»  
I семестр

Задание 4

«Процедуры и функции в качестве параметров»

Студент: Касумова Н.Р.

Группа: М8О-103Б-22

Руководитель: С.П.Никулин

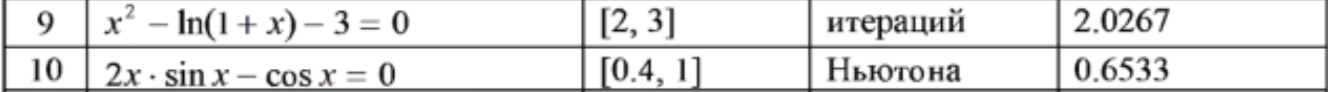
Оценка:

Подпись преподавателя:

**Задание**

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итерации, 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, начиная с варианта с заданным номером. Если метод неприменим, дать математическое обоснование и графическую иллюстрацию.

**Вариант №9 и №10**



**Общие сведения о программе**

Аппаратное обеспечение: домашний ноутбук

Операционная система: macOS

Язык и система программирования: C

Компиляция в консоли: gcc kp4.c

Вызов программы: ./a.out

**Описание методов:**

1. **Метод дихотомии**

Численное нахождение приближенного значения корня функции строится на базе следствия из теоремы Больцано-Коши: «Если непрерывная функция на концах некоторого интервала имеет значение разных знаков, то внутри этого интервала у нее есть как минимум один корень».

Задача заключается в том, чтобы найти корень методом половинного деления, т.е. найти приближенное значение корня с заданной точностью .

Пусть функция непрерывна на отрезке , – единственный корень уравнения .

Поделим отрезок пополам. Получим точку и два отрезка .

Если , то корень найден (.

Если нет, то из двух полученных отрезков надо выбрать один такой, что ; , если или , если

Новый отрезок делим пополам. Получаем середину этого отрезка и так далее.

Для того, чтобы найти приближенное значение корня с точностью до , необходимо остановить процесс половинного деления на таком шаге , на котором и вычислить . Тогда можно взять .

Функция непрерывна на отрезке и на концах его принимает значения одинаковых знаков: ;

Функция непрерывна на отрезке и на концах его принимает значения разных знаков: ;

1. **Метод итераций**

Суть метода в поиске по известному приближению искомой величины следующего, более точного приближения.

Пусть дана функция . Заменим исходное уравнение на эквивалентное . Выберем начальное приближение корня . Тогда получим некоторое число . Теперь подставляя вместо число получим . Повторяя этот процесс, будем иметь последовательность . Если эта последовательность сходящаяся, то существует предел , то данный предел является корнем уравнения и может быть вычислен по формуле .

Условие сходимости метода итераций:

Найдем для функции ;

Условие сходимости выполняется.

Найдем для функции

Условие сходимости выполняется.

1. **Метод Ньютона**

Метод Ньютона – частный случай метода итераций (). Суть метода состоит в разбиении отрезка на два отрезка с помощью касательной и выборе нового отрезка от точки пересечения касательной с осью абсцисс до неподвижной точки, на которой функция меняет знак и содержит решение.

Метод итераций сходится тогда и только тогда, когда , подставим в условие выражение для и получим условие сходимости метода Ньютона:

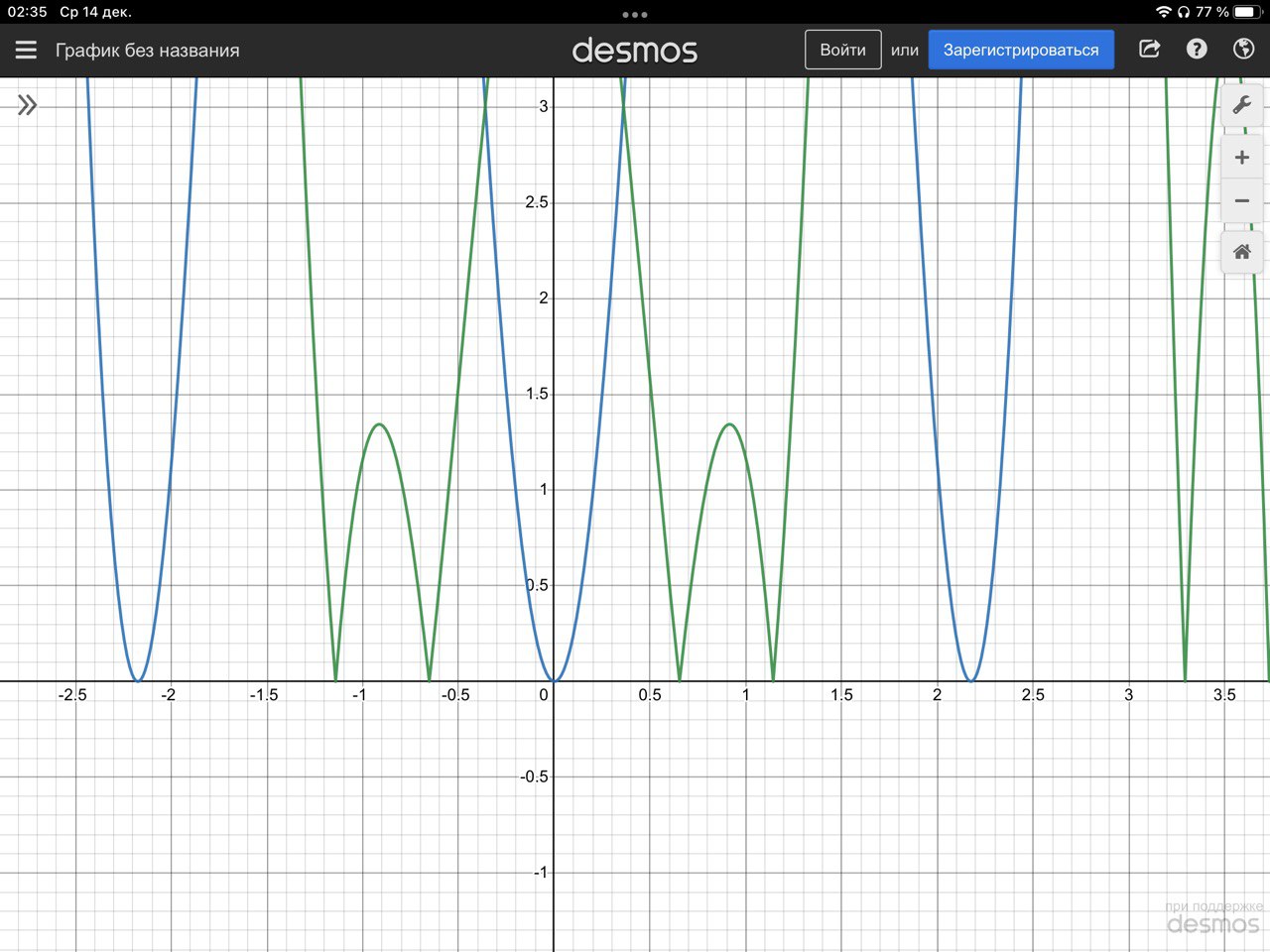
Условие окончания: если , то значение считается приближенным значением корня уравнения

**Поиск производных**

Вариант 9:

| на отрезке

Вариант 10:

| на отрезке

**Описание переменных и функций**

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| a | double | Левая граница отрезка |
| b | Правая граница отрезка |
| eps | Машинное эпсилон |
| func\_1(2) | Заданная функция 9(10) варианта |
| diff1\_1(2) | Производная функции |
| funcit\_1(2) | Функция в виде x = f(x) |
| diff2\_1(2) | Вторая производная функции |
| dichotomy | Метод дихотомии |
| iteration | double | Метод итераций |
| newton | Метод Ньютона |

**Протокол**

#include <stdio.h>

#include <math.h>

double ep(void) {

double e = 1.0;

while ((1 + (e / 2.0))>1)

e /= 2.0;

return e;

}

double func\_1(double x) {

return 2\*x\*sin(x) - cos(x);

}

double func\_2(double x) {

return x\*x-logl(1+x)-3;

}

double funcit\_1(double x) {

return (cos(x))/(2\*sin(x));

}

double funcit\_1\_1(double x){

return asin(sin(x));

}

double funcit\_2(double x) {

return sqrt(logl(1+x)+3);

}

double funcitdiff\_1(double x) {

return (-2\*pow(sin(x),2) - 2\*pow(cos(x),2))/(4\*pow(sin(x),2));

}

double funcitdiff\_2(double x) {

return 1/(2\*(x+1)\*sqrt(logl(1+x)+3));

}

double diff1\_1(double x) {

return 2\*sin(x) + 2\*x\*cos(x) + sin(x);

}

double diff1\_2(double x) {

return 2\*x-1/(x+1);

}

double diff2\_1(double x) {

return 5\*cos(x) - 2\*x\*sin(x);

}

double diff2\_2(double x) {

return 2+1/pow((x+1),2);

}

double dichotomy(double(\*F)(double), double k,double l) {

double a = k,b = l, eps = ep() \* 100;

while (fabs(a - b)>eps) {

if ((F(a)\*F((a + b) / 2))>0) a = (a + b) / 2;

else if ((F(b)\*F((a + b) / 2))>0) b = (a + b) / 2;

}

return (a + b) / 2;

}

double iteration(double(\*F1)(double), double(\*F2)(double),double k,double l) {

double a = k, b = l;

if (fabs(F2(a))>1 || fabs(F2(b))>1)

return 0;

else {

double eps = ep() \* 100;

double x = (a + b) / 2.0;

while (fabs(F1(x) - x) > eps)

x = F1(x);

return F1(x);

}

}

double newton(double(\*F1)(double), double(\*F2)(double), double(\*F3)(double),double k,double l) {

double a = k, b = l;

if (fabs(F1(a)\*F3(a))<(F1(a)) || fabs(F1(b)\*F3(b))<(F1(b)))

return 0;

double eps = ep() \* 100;

double x = (a + b) / 2.0;

while (fabs((x - F1(x) / F2(x)) - x) > eps)

x = x - F1(x) / F2(x);

return (x - F1(x) / F2(x));

}

int main() {

printf("eps = %.21lf\n", ep());

printf("-----------------------------------------------------------------------------------------------------\n");

printf("| Уравнения |Метод дихотомии|Метод итераций| Метод Ньютона | Прибл. значение корня | Отрезок |\n");

printf("-----------------------------------------------------------------------------------------------------\n");

printf("|2x\*sin(x) - cos(x)|%.13lf|", dichotomy(func\_1,0.4,1));

if ((iteration(funcit\_1, funcitdiff\_1, 0.4,1) == 0) && (iteration(funcit\_1\_1,funcitdiff\_1,0.4,1)==0))

printf(" Невозможно |"); else

printf("%.12lf|", iteration(funcit\_1, funcitdiff\_1,0.4,1)); if (newton(func\_1, diff2\_1, diff1\_1,0.4,1) == 0)

printf(" Невозможно |\n"); else

printf("%.13lf| 0.6533 | [0.4,1] |\n", newton(func\_1, diff2\_1, diff1\_1,0.4,1));

printf("-----------------------------------------------------------------------------------------------------\n");

printf("|x^2 - ln(1+x) - 3 |%.13lf|", dichotomy(func\_2,2,3));

if (iteration(funcit\_2, funcitdiff\_2,2,3) == 0)

printf(" Невозможно |"); else

printf("%.12lf|", iteration(funcit\_2, funcitdiff\_2,2,3)); if (newton(func\_2, diff2\_2, diff1\_2,2,3) == 0)

printf(" Невозможно |\n"); else

printf("%.13lf| 2.0267 | [2,3] |\n", newton(func\_2, diff2\_2, diff1\_2,2,3));

printf("-----------------------------------------------------------------------------------------------------\n");

}

**Результат работы программы**

Изображение выглядит как текст

Автоматически созданное описание

n@MacBook-Pro-N doc % gcc kp4.c

n@MacBook-Pro-N doc % ./a.out

eps = 0.000000000000000222045

-----------------------------------------------------------------------------------------------------

| Уравнения |Метод дихотомии|Метод итераций| Метод Ньютона | Прибл. значение корня | Отрезок |

-----------------------------------------------------------------------------------------------------

|2x\*sin(x) - cos(x)|0.6532711870944| Невозможно |0.6532711870944| 0.6533 | [0.4,1] |

-----------------------------------------------------------------------------------------------------

|x^2 - ln(1+x) - 3 |2.0266892632435|2.026689263244|2.0266892632435| 2.0267 | [2,3] |

-----------------------------------------------------------------------------------------------------

n@MacBook-Pro-N doc %

**Выводы**

Численные методы являются основным инструментом решения современных прикладных задач. Аналитическое решение различных задачи в виде формульных соотношений является скорее исключением, нежели правилом в силу сложного и приближенного характера исследуемых моделей. Поэтому численный анализ математических моделей является в настоящее время актуальным и наиболее эффективным аппаратом конструктивного исследования прикладных проблем.

В курсовом проекте были рассмотрены 3 численных метода решения уравнений: метод дихотомии, метод итераций и метод Ньютона. В процессе написания программы было изучено аналитические условия применимости данных методов и их реализация в коде. Были составлены форматы ввода и вывода, проведены тесты программы на разных значениях точности и их расхождения с фактическим значением корня.