|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  федеральное государственное автономное образовательное учреждение высшего образования  «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ» |

компьютерных технологий и программной инженерии

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ОТЧЁТ ПО ПРАКТИКЕ  ЗАЩИЩЁН С ОЦЕНКОЙ  Руководитель |  | | | |
| старший преподаватель |  |  |  | А.А. Фоменкова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ОТЧЁТ ПО ПРАКТИКЕ

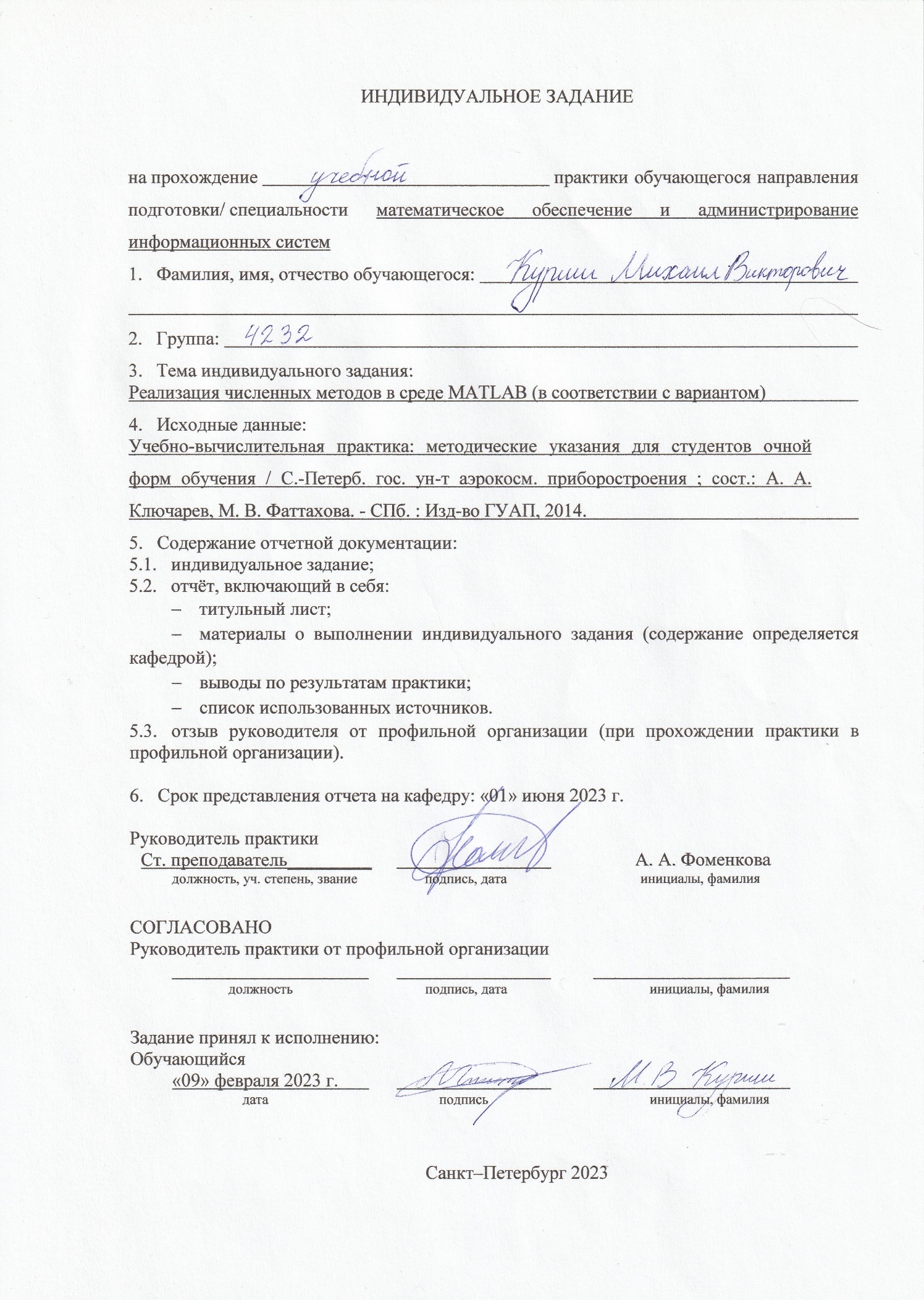
|  |  |  |
| --- | --- | --- |
| вид практики | учебная | |
| тип практики |  | |
| на тему индивидуального задания | | Реализация численных методов в среде MATLAB (в |
| соответствии с вариантом) | | | |
|  | | | |

|  |  |
| --- | --- |
| выполнен | Куриш Михаилом Викторовичем |
| фамилия, имя, отчество обучающегося в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.04 |  | «Программная инженерия» |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности |  |  | «Проектирование программных систем» |
|  | код |  | наименование направленности |
|  | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Обучающийся группы № | 4232 |  | C:\Users\Home1\Documents\IMG_20230513_0003.jpg |  | М. В. Куриш |
|  | номер |  | подпись, дата |  | инициалы, фамилия |

Санкт–Петербург 2023



**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc134191449)

[ОСНОВНАЯ ЧАСТЬ ОТЧЕТА 6](#_Toc134191450)

[1 Раздел 1. Решение нелинейных и трансцендентных уравнений 6](#_Toc134191451)

[1.1 Математическая постановка задачи 6](#_Toc134191452)

[1.2 Описание метода численной реализации задания 7](#_Toc134191453)

[1.3 Описание ручного счета тестового примера предложенным методом 8](#_Toc134191454)

[1.4 Разработка алгоритмов решения задачи 11](#_Toc134191455)

[1.5 Листинг программы 12](#_Toc134191456)

[1.6 Результат работы программы 14](#_Toc134191457)

[1.7 Результат реализации метода с помощью встроенных функций 18](#_Toc134191458)

[2. Раздел 2. Интерполяция 19](#_Toc134191459)

[2.1 Математическая постановка задачи 19](#_Toc134191460)

[2.2 Описание метода численной реализации задания 20](#_Toc134191461)

[2.3 Описание ручного счета тестового примера предложенным методом 22](#_Toc134191462)

[2.4 Разработка алгоритмов решения задачи 23](#_Toc134191463)

[2.5 Листинг программы 24](#_Toc134191464)

[2.6 Результат работы программы 28](#_Toc134191465)

[2.7 Результат реализации метода с помощью встроенных функций 32](#_Toc134191466)

[3. Раздел 3. Численное интегрирование 33](#_Toc134191467)

[3.1 Математическая постановка задачи 33](#_Toc134191468)

[3.2 Описание метода численной реализации задания 34](#_Toc134191469)

[3.3 Описание ручного счета тестового примера предложенным методом 35](#_Toc134191470)

[3.4 Разработка алгоритмов решения задачи 37](#_Toc134191471)

[3.5 Листинг программы 38](#_Toc134191472)

[3.6 Результат работы программы 40](#_Toc134191473)

[3.7 Результат реализации метода с помощью встроенных функций 43](#_Toc134191474)

[4 Раздел 4. Обработка ошибок 44](#_Toc134191475)

[ЗАКЛЮЧЕНИЕ 46](#_Toc134191476)

[Решение нелинейных и трансцендентных уравнений 46](#_Toc134191477)

[Интерполяция 46](#_Toc134191478)

[Численное интегрирование 46](#_Toc134191479)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 48](#_Toc134191480)

[ПРИЛОЖЕНИЯ 49](#_Toc134191481)

# **ВВЕДЕНИЕ**

Данный отчет представляет решение задачи, поставленной учебной практикой, а именно: реализована программа с текстовым интерфейсом, позволяющая решать нелинейные уравнения, интерполировать таблично заданную функцию и интегрировать. Согласно варианту №8, в указанных математических операциях используются метод Ньютона, метод Лагранжа и метод трапеций соответственно.

Цели работы:

* Закрепление теоретических знаний, полученных студентами на лекционных, лабораторных и практических занятиях;
* Развитие навыков самостоятельной работы по анализу информационных материалов печатных и электронных источников;
* Закрепление навыков программирования на языке MATLAB;
* Освоение работы с функциями и массивами данных, получение навыков построения интерфейса пользователя.

# **ОСНОВНАЯ ЧАСТЬ ОТЧЕТА**

## **1 Раздел 1. Решение нелинейных и трансцендентных уравнений**

### **1.1 Математическая постановка задачи**

Решить уравнение методом Ньютона (касательных). Для отделения корней использовать графический метод.

Согласно варианту №8, .

### **1.2 Описание метода численной реализации задания**

Имея функцию и определив по графику отрезок, на котором будет выполняться поиск корня (будем считать, что функция на этом отрезке непрерывна и монотонна, поскольку проверка этих характеристик с помощью ПК возможна только методом перебора и даже при большом разбиении отрезка не гарантирует абсолютную точность), надо проверить отрезок на возможность применения к нему метода Ньютона [1]. Для этого ищется вторая производная функции . Границы промежутка (на котором ф-я монотонна и непрерывна) должны иметь разный знак, чтобы имелся ровно один корень, т.е. . Кроме того, функция должна иметь одинаковую выпуклость на всем промежутке, что означает, , если условие для данного конца отрезка выполнено, он берется за начальное приближение, иначе (учитывая все уже наложенные ограничения на промежуток) за начальное приближение берется второй конец отрезка (для него будет справедливо ).

Пусть начальным приближением является . Рассчитаем точность поиска корня, когда за корень принято начальное приближение: . Здесь – значение функции в точке , а – значение первой производной функции в этой же точке.

Если полученная погрешность слишком велика, необходимо перейти к следующему приближению . Аналогично предыдущему пункту, найдем точность для : .

Поиск нового приближения повторяется до тех пор, пока не будет достигнута достаточная точность вычисления.

### **1.3 Описание ручного счета тестового примера предложенным методом**

Согласно варианту, .

Выражение принимает вид .

Функция же: .

Пусть необходимая точность .

Для поиска корня с использованием графического метода необходимо построить график (Рисунок 1).

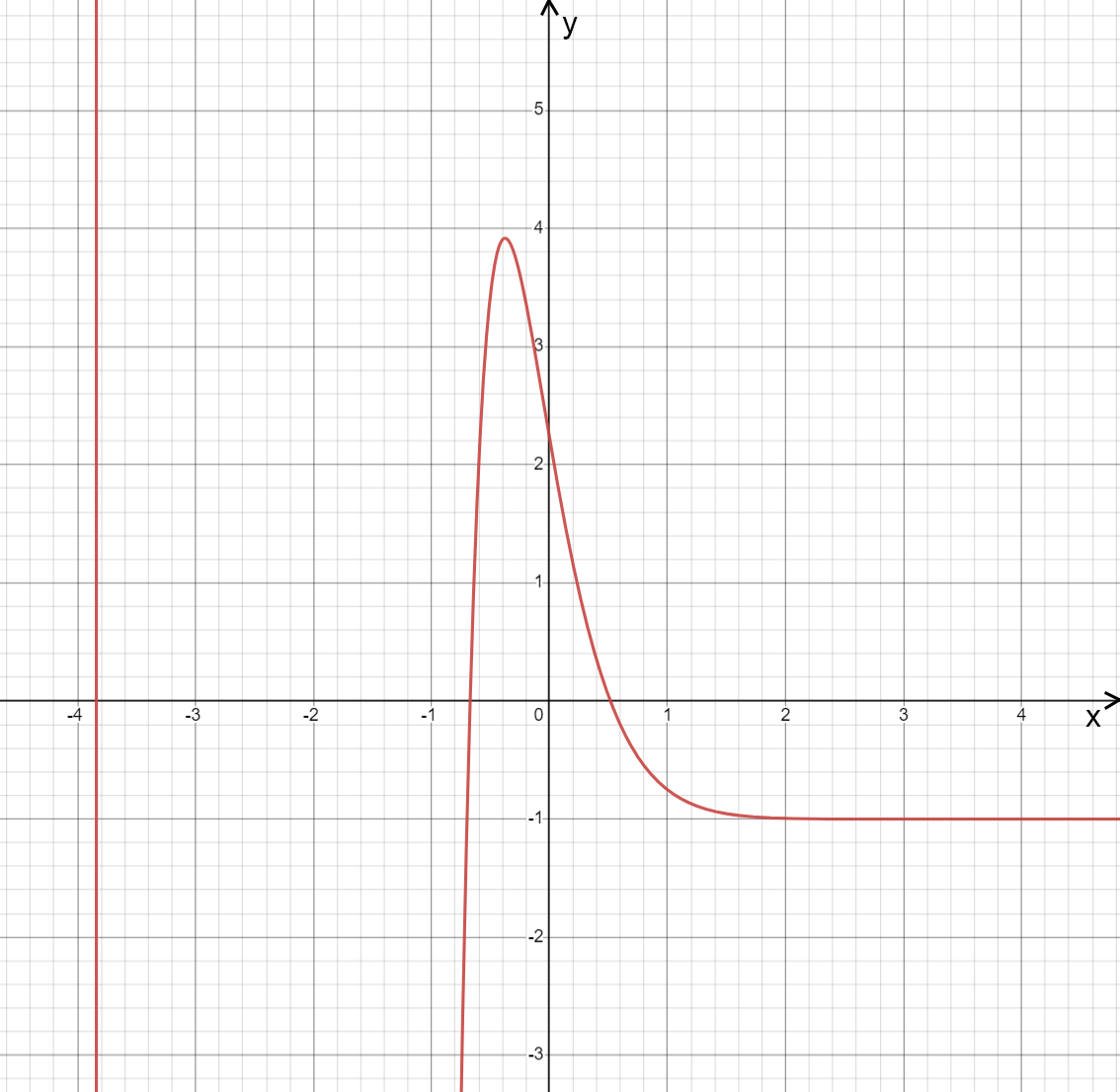
****

Рисунок 1 - График функции по варианту для ручного счета

Анализируя график (Рисунок 1), выберем промежуток . На этом промежутке функция непрерывна, монотонна.

Для последующих расчетов найдем первую и вторую производные функции:

Проверим концы промежутка на разность знаков.

Проверим левый конец промежутка на выпуклость:

Шаг 1.

Исходя написанного выше, за начальное приближение возьмем . Найдем точность вычисления для :

Точность не удовлетворяет заданной ().

Шаг 2.

Следующее приближение:

Точность вычисления для :

Точность не удовлетворяет заданной ().

Шаг 3.

Следующее приближение:

Точность вычисления для :

Точность не удовлетворяет заданной ().

Шаг 3.

Следующее приближение:

Точность вычисления для :

Точность удовлетворяет заданной ().

Ответ: для функции на промежутке существует корень (с погрешностью ).

### **1.4 Разработка алгоритмов решения задачи**

Алгоритм решения задачи представлен в виде блок-схемы [2] на рисунке 2.

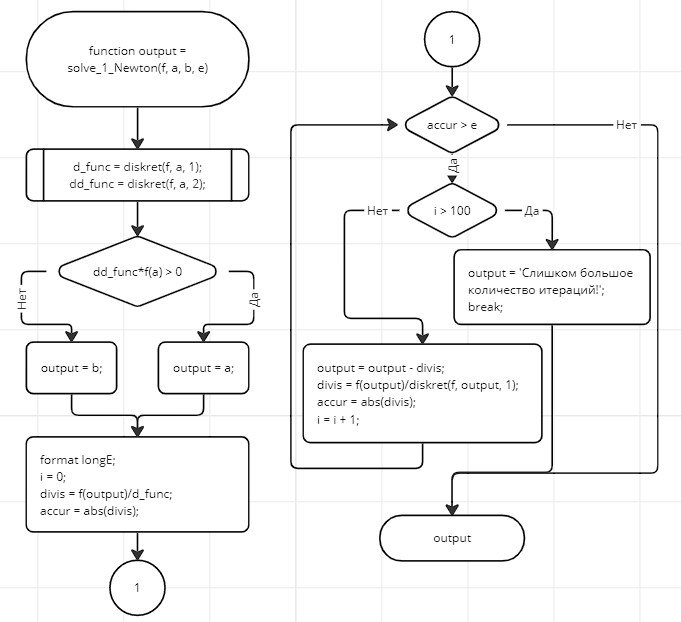
****

Рисунок 2 - Блок-схема функции “solve\_1\_Newton”

### **1.5 Листинг программы**

Реализация математического алгоритма с помощью MatLab состоит из двух функций: “solve\_1\_Newton” и “diskret”.

Функция “solve\_1\_Newton” является основной функцией, её код приведён в листинге 1.

*Листинг 1 - Функция “solve\_1\_Newton”*

|  |
| --- |
| function output = solve\_1\_Newton(f, a, b, e)  % Функция, реализующая поиск корня на интервале функции по  % методу Ньютона.  % f - function handle, функция  % a, b - левая и правая границы промежутка  % e - погрешность вычисления  %  % Результат - вычисленное значение корня или текст возникшей ошибки  d\_func = diskret(f, a, 1); % Определение производной в точке а  dd\_func = diskret(f, a, 2); % Определение второй производной  if dd\_func\*f(a) > 0 % Условие для точки а  output = a;  else  output = b;  end  format longE;  i = 0; % Счетчик количества итераций  divis = f(output)/d\_func; % Переменная, хранящее изменение функции в данной точке  accur = abs(divis); % Точность определения корня при данном значении корня  while accur > e % Пока точность недостаточна, выполнить алгоритм Ньютона  if i > 100  output = 'Слишком большое количество итераций!';  break;  end  output = output - divis; % Новый корень  divis = f(output)/diskret(f, output, 1);  accur = abs(divis); % Точность при новом корне  i = i + 1;  end  end |

Функция “diskret” нужна для получения значения первой и второй производной функции в данной точке, её код приведён в листинге 2.

*Листинг 2 - Функция “diskret”*

|  |
| --- |
| function rez = diskret(f, x, counter)  % Функция, определяющая производную (первую или вторую,  % в зависимости от параметра counter) функции f в точке x.  %  % При counter == 1 - первая производная  % При counter ~= 1 - вторая производная  %  % Результат - числовое значение соответствующей производной  if counter == 1  rez = (f(x + 0.0001) - f(x))/0.0001;  else  rez = (f(x + 0.0002) - 2\*f(x + 0.0001) + f(x))/0.00000001;  end |

### **1.6 Результат работы программы**

На рисунке 3 изображено главное меню программы.

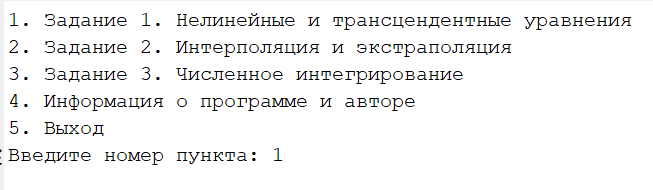


Рисунок 3 - Главное меню программы, выбор 1 подменю

На рисунке 4 изображено подменю задания 1 программы.

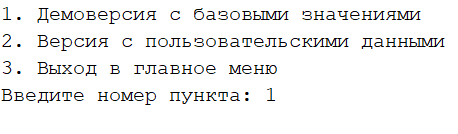


Рисунок 4 - Подменю 1, выбор версии решения

На рисунке 5 изображен результат работы демонстрационного варианта первого задания.

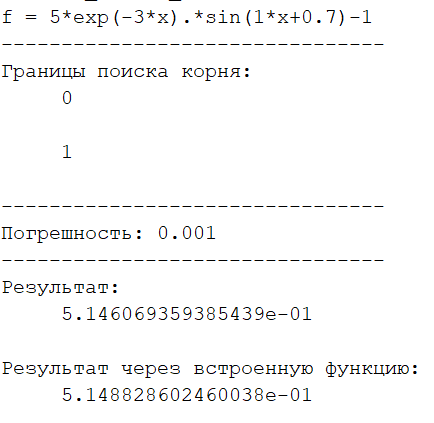


Рисунок 5 - Результат работы демоварианта 1 раздела, консоль

На рисунке 6 изображен график демоварианта первого задания.

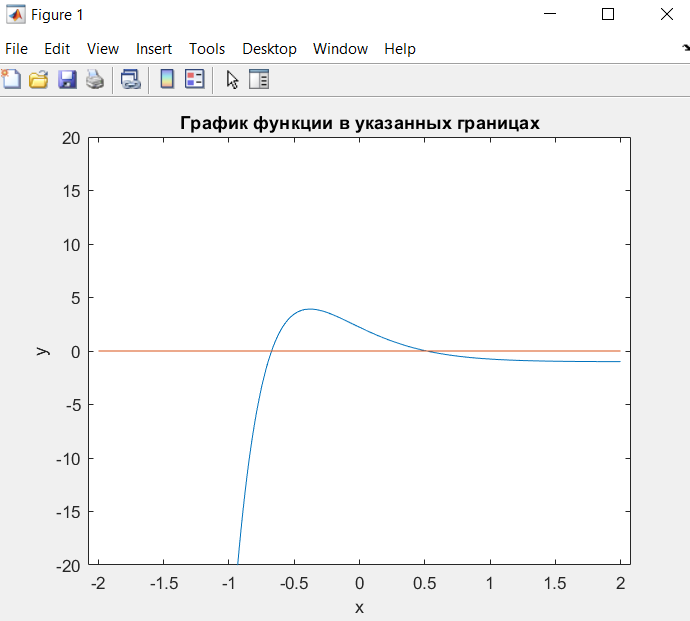


Рисунок 6 - Результат работы демоварианта 1 раздела, график

На рисунке 7 изображен результат работы варианта с пользовательскими данными первого задания.

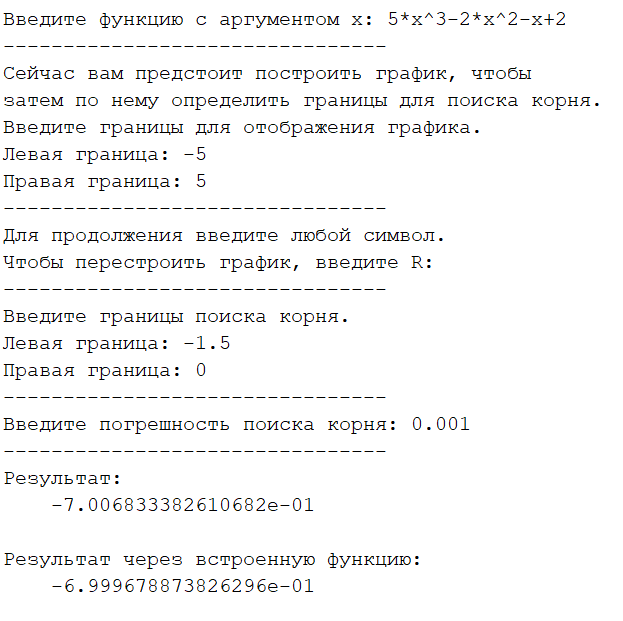


Рисунок 7 - Результат работы варианта с пользовательскими данными 1 раздела, консоль

На рисунке 8 изображен график варианта с пользовательскими данными первого задания.

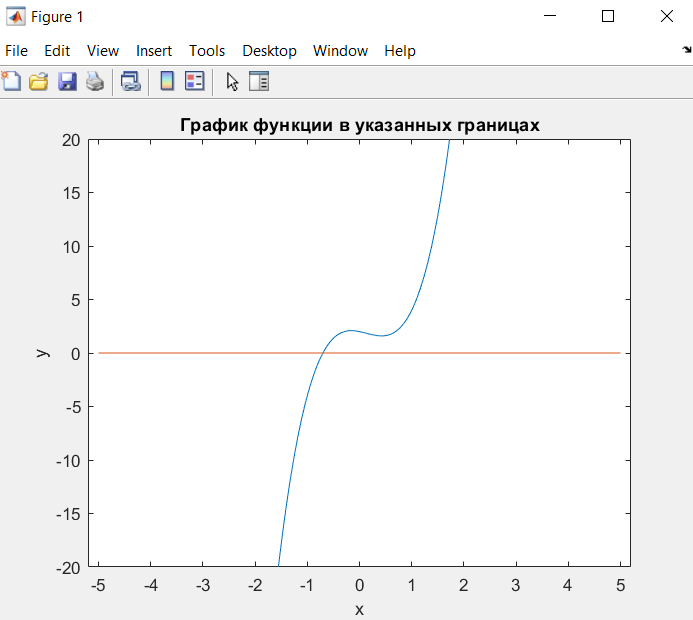


Рисунок 8 - Результат работы варианта с пользовательскими данными 1 раздела, график

### **1.7 Результат реализации метода с помощью встроенных функций**

Для решения поставленной задачи, учитывая использование метода Ньютона, должна использоваться встроенная функция “fzero”. Результат работы продемонстрирован на рисунках 5 (демовариант) и 7 (пользовательские значения).

## **2. Раздел 2. Интерполяция**

### **2.1 Математическая постановка задачи**

Для заданной таблично функции найти значения функции в точках: и с помощью интерполяционного многочлена Лагранжа.

Согласно варианту №8, функция задана таблицой 1.

Таблица 1 - Таблично заданная функция

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1,2 | 1,5 | 1,7 | 2,0 |
|  | 7,277 | 7,545 | 7,811 | 8,089 |

### **2.2 Описание метода численной реализации задания**

Метод интерполяционного полинома Лагранжа реализуется с помощью формул 1 и 2 [1]:

|  |  |
| --- | --- |
|  | (1) |

|  |  |
| --- | --- |
|  | (2) |

В данных формулах , , – элементы заданной таблицы точек.

Разобъем вычисление полинома на вычисление суммы результатов другой функции (произведений). Будем работать с полиномами как с векторами коэффициентов при степенях аргумента.

Разобъем дробь на . Элемент можно представить как , а элемент как . Соответственно, в условленном виде, полином при подстановке значений и выглядел бы: , где – коэффициент при , а – коэффициент при .

Произведение полиномов можно реализовать следующим образом, по правилам математики (формула 3).

|  |  |
| --- | --- |
|  | (3) |

Для каждого i-го элемента (коэффициента) одного вектора необходимо искать произведение с j-м элементом второго вектора, причем полученное произведение будет коэффициентом при .

Просуммировав произведения (полиномы), получим конечный полином.

Суммирование реализуется также по законам математики (формула 4).

|  |  |
| --- | --- |
|  | (4) |

В формуле (4) подразумевается, что n больше чем m на k.

Стоит заметить, что в MatLab вектора индексируются не с 0, а с 1, а также вектор читается справа-налево, поэтому для удобства вместо записи будем пользоваться записью , а вместо формул (3) и (4) будем использовать формулы (5) и (6) соответственно.

|  |  |
| --- | --- |
|  | (5) |

|  |  |
| --- | --- |
|  | (6) |

Пройдя по таблице точек, получим вектор коэффициентов искомого полинома. В заключение следует преобразовать вектор коэффициентов в полином (функцию). При ручном расчете это можно сделать приписывая t-му по счету (при индексиции как в MatLab, т.е. начиная с 1) элементу. В компьютерной реализации эту задачу реализуем отдельной функцией coefs2func (выходные данные – function handle).

### **2.3 Описание ручного счета тестового примера предложенным методом**

При ручном счете не будем делать промежуточное действие - представления полиномов в виде векторов коэффициентов.

Функция задана таблицой 1 (см. пункт 2.1).

Найти значение функции в точке .

По формуле (2):

По формуле (1):

Найдем значение для :

Ответ: найден интерполяционный полином , найдено промежуточное значение для .

### **2.4 Разработка алгоритмов решения задачи**

Алгоритм решения задачи представлен в виде блок-схемы на рисунке 9.

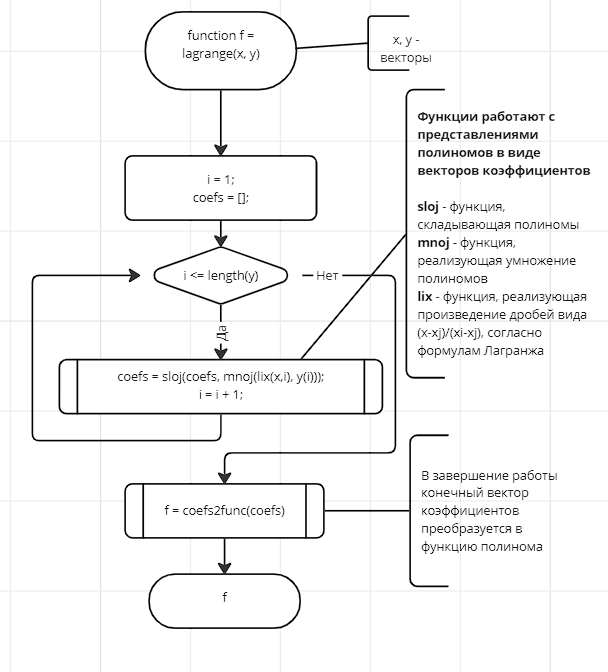


Рисунок 9 - Блок-схема функции “lagrange”

### **2.5 Листинг программы**

Реализация математического алгоритма с помощью MatLab состоит из нескольких функций: “lagrange”, “sloj”, “mnoj”, “lix”, “coefs2func”.

Функция “ lagrange” является основной функцией, её код приведён в листинге 3.

*Листинг 3 - Функция “lagrange”*

|  |
| --- |
| function f = lagrange(x, y)  % Функция, реализующая построение функции по  % методу интерполяционного полинома Лагранжа.  %  % x - вектор аргументов  % y - вектор значений для соответствующих аргументов  %  % Результат - function handle, функция  i = 1;  coefs = []; % Создание вектора коэффициентов полинома  while (i <= length(y))  coefs = sloj(coefs, mnoj(lix(x,i), y(i))); % Суммирование произведений значений (y(i)) с соответствующими произведениями дробей (lix(x,i))  i = i + 1;  end  f = coefs2func(coefs); % Преобразование вектора коэффициентов в function handle  end |

Функция “sloj” выполняет сложение двух полиномов, представленных в виде векторов коэффициентов, её код приведён в листинге 4.

*Листинг 4 - Функция “sloj”*

|  |
| --- |
| function plnm = sloj(lft, rght)  % Функция, реализующая сложение полиномов, представленных в виде  % векторов коэффициентов (разряды келичиваются слева-направо).  %  % lft - вектор коэфф. первого полинома  % rght - вектор коэфф. второго полинома  %  % Результат - вектор коэффициентов полинома  plnm = zeros(1, mymax([length(lft), length(rght)])); % Создание вектора коэффициентов итогового полинома  i = 1;  while (i <= length(lft))  plnm(i) = plnm(i) + lft(i); % Внесение в итоговый вектор значений первого вектора  i = i + 1;  end  i = 1;  while (i <= length(rght))  plnm(i) = plnm(i) + rght(i); % Сложение с имеющимися значениями итогового вектора значений второго вектора  i = i + 1;  end  end |

Функция “mnoj” выполняет перемножение двух полиномов, представленных в виде векторов коэффициентов, её код приведён в листинге 5.

*Листинг 5 - Функция “mnoj”*

|  |
| --- |
| function plnm = mnoj(lft, rght)  % Функция, реализующая умножение полиномов, представленных в виде  % векторов коэффициентов (разряды келичиваются слева-направо).  %  % lft - вектор коэфф. первого полинома  % rght - вектор коэфф. второго полинома  %  % Результат - вектор коэффициентов полинома  plnm = zeros(1, length(lft)+length(rght)-1); % Создание вектора коэффициентов итогового полинома  i = 1;  while (i <= length(rght))  j = 1;  while (j <= length(lft))  plnm(i+j-1) = plnm(i+j-1) + lft(j) \* rght(i); % Заполнение итогового вектора коэффициентов согласно правилам математики  j = j + 1;  end  i = i + 1;  end  end |

Функция “lix” реализует нахождение элемента в соответствии с формулой 2 (см. пункт 2.2), её код приведён в листинге 6.

*Листинг 6 - Функция “lix”*

|  |
| --- |
| function coefs = lix(x, i)  % Функция, реализующая произведения всех дробей вида (x-xj)/(xi-xj)  % по вектору x для определенного i соответственно  % методу интерполяционного полинома Лагранжа.  %  % Вычисления происходят с векторами коэффициетов полиномов.  % Конечный вектор - есть вектор коэффициентов полинома, являющегося  % произведением всех дробей, описанных выше.  %  % x - вектор аргументов  % i - значение индекса i в формуле произведений дробей по Лагранжу  %  % Результат - вектор коэффициентов  j = 1;  coefs = 1; % Создание начального вектора коэффициентов  while (j <= length(x))  if (j ~= i)  coefs = mnoj(coefs, [-x(j)/(x(i)-x(j)), 1/(x(i)-x(j))]); % Умножение вектора коэффициентов (полинома) на очередную дробь по формуле Лагранжа  end  j = j + 1;  end  end |

Функция “coefs2func” формирует function handle из вектора коэффициентов полинома, её код приведён в листинге 7.

*Листинг 7 - Функция “coefs2func”*

|  |
| --- |
| function f = coefs2func(coefs)  % Функция, формирующая Function Handle из вектора коэффициентов полинома.  %  % Коэффициенты распологаются в порядке от коэфф. младшего разряда (слева)  % до старшего (справа).  %  % Результат - функция, т.е. Function Handle  s = ''; % Создание строки, в которую будет формироваться функция из вектора коэффициентов  i = 1;  while (i <= length(coefs)) % Обработка коэффициентов  if (coefs(i) > 0)  if (coefs(i) == 1)  s = strcat('+x.^', num2str(i-1, 10), s);  else  s = strcat('+', num2str(coefs(i), 10), '\*x.^', num2str(i-1, 10), s);  end  else  if (coefs(i) < 0)  if (coefs(i) == -1)  s = strcat('-x.^', num2str(i-1, 10), s);  else  s = strcat(num2str(coefs(i), 10), '\*x.^', num2str(i-1, 10), s);  end  end  end  i = i + 1;  end  f = eval(strcat('@(x)', s)); % Преобразование полученной строки в Function Handle  end |

### **2.6 Результат работы программы**

На рисунке 10 изображено подменю задания 2 программы.

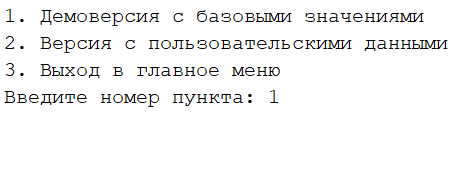


Рисунок 10 - Подменю 2, выбор версии решения

На рисунке 11 изображен результат работы демонстрационного варианта второго задания в консоли.

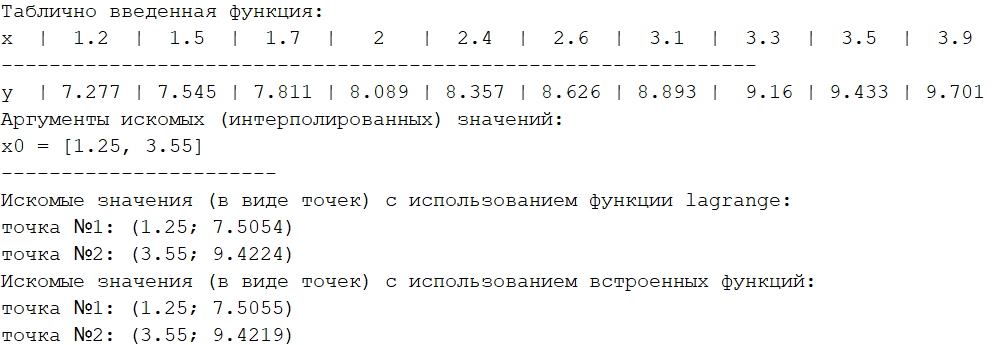


Рисунок 11 - Результат работы демоварианта 2 раздела, консоль

На рисунке 12 изображен результат работы демонстрационного варианта второго задания на графике с использованием разработанной функции “lagrange”.

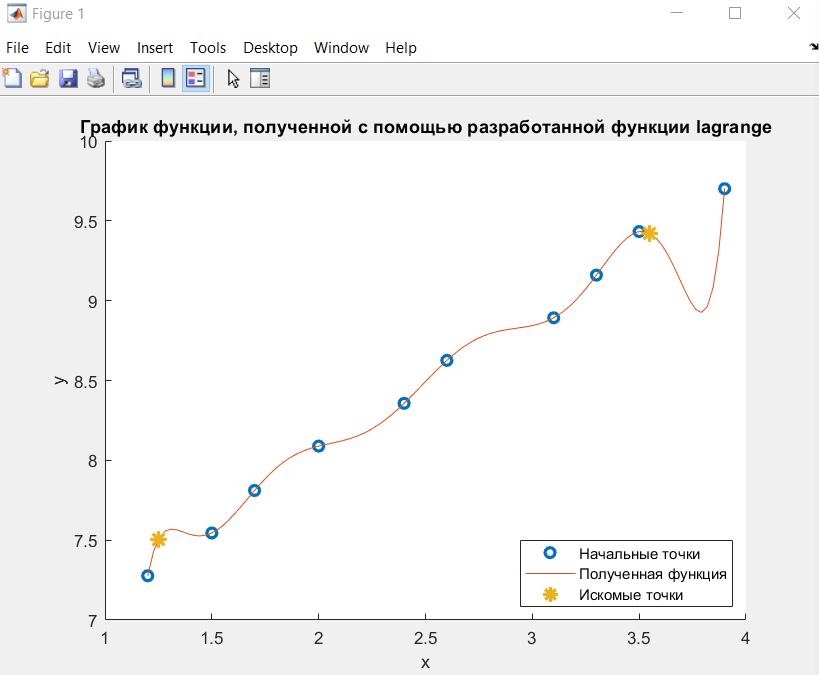


Рисунок 12 - Результат работы демоварианта 2 раздела, график, “lagrange”

На рисунке 13 изображен результат работы демонстрационного варианта второго задания на графике с использованием встроенных функций.

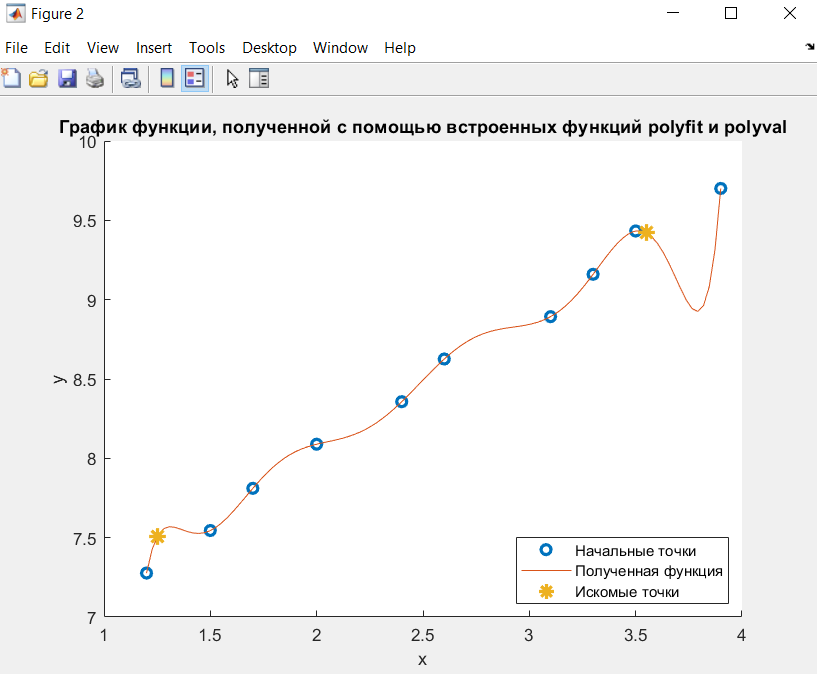


Рисунок 13 - Результат работы демоварианта 2 раздела, график, встроенные функции

На рисунке 14 изображен результат работы варианта с пользовательскими данными второго задания в консоли.

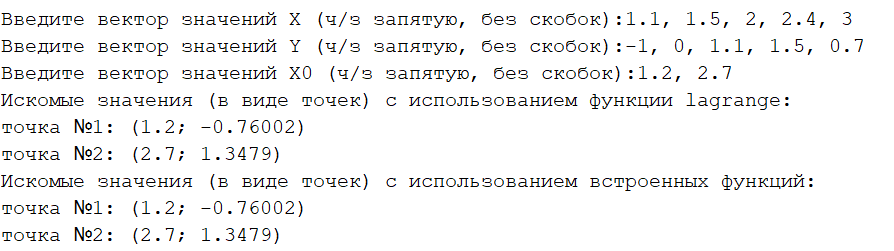


Рисунок 14 - Результат работы варианта с пользовательскими данными 2 раздела, консоль

На рисунке 15 изображен результат работы варианта с пользовательскими данными второго задания на графике с использованием разработанной функции “lagrange”.

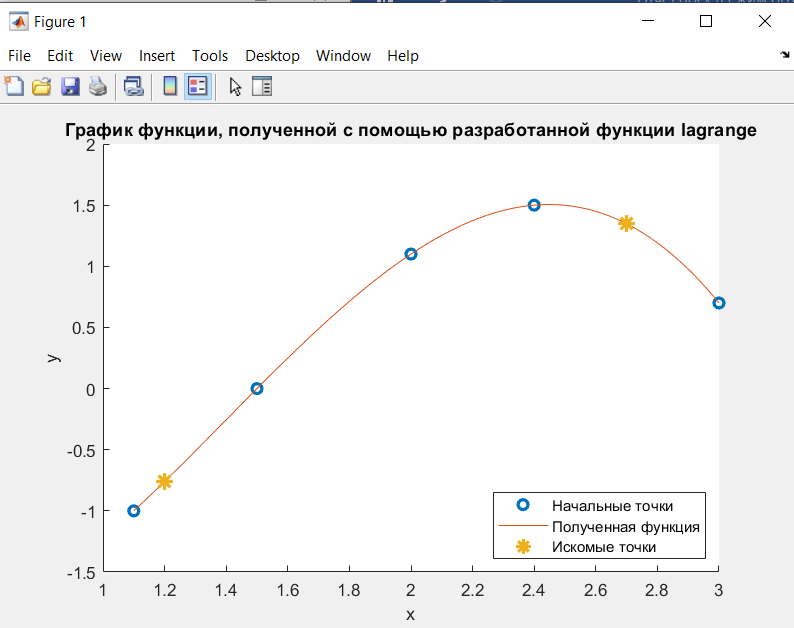


Рисунок 15 - Результат работы варианта с пользовательскими данными 2 раздела, график, “lagrange”

На рисунке 16 изображен результат работы варианта с пользовательскими данными второго задания на графике с использованием встроенных функций.

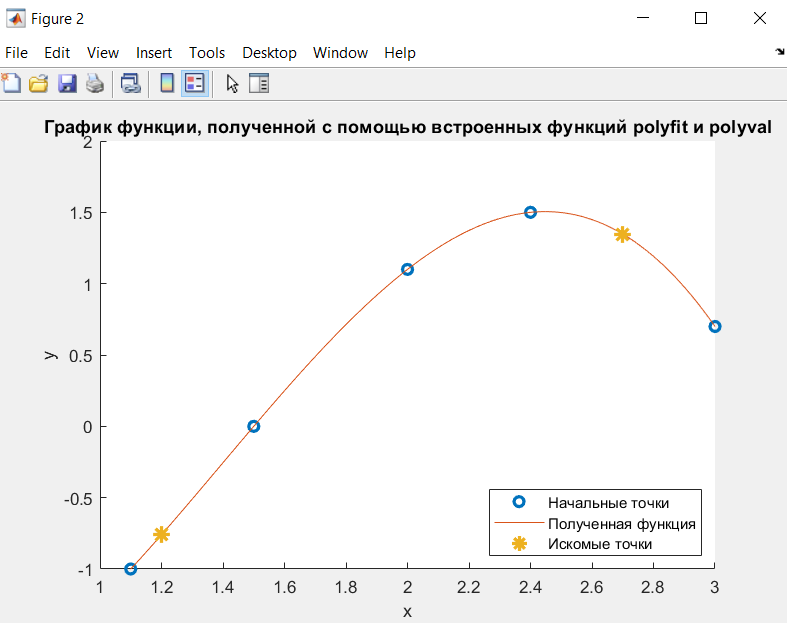


Рисунок 16 - Результат работы варианта с пользовательскими данными 2 раздела, график, встроенные функции

### **2.7 Результат реализации метода с помощью встроенных функций**

Для решения поставленной задачи, учитывая использование метода Лагранжа, должна использоваться встроенная функция “polyfit” совместно с “polyval” [2]. Результат работы продемонстрирован на рисунках 11, 13 (демовариант) и 14, 16 (пользовательские значения).

## **3. Раздел 3. Численное интегрирование**

### **3.1 Математическая постановка задачи**

Вычислите значение определенного интеграла методом трапеций. Оцените погрешность вычислений.

Согласно варианту №8, .

### **3.2 Описание метода численной реализации задания**

Определенный интеграл – это площадь под графиком подынтегральной функции. Эту площадь можно найти, не прибегая к вычислению непосредственно интеграла, как это и позволяет сделать метод трапеций. В данном методе промежуток интегрирования разбивается точками с шагом . Точки , , , образуют трапецию, высота которой равна , а основания равны и . Метод трапеций заключается в нахождении площадей всех трапеций, образованных в интегрируемом промежутке и сложении их для приближенного нахождения площади под графиком функции. Площадь трапеции ищется по формуле (7).

|  |  |
| --- | --- |
|  | (7) |

Значение же интеграла ищется с помощью формулы (8), где n – количество точек, на которые разбит промежуток интегрирования.

|  |  |
| --- | --- |
|  | (8) |

Для оценки погрешности при использовании метода трапеций пользуются формулой (9), где – значение интеграла, вычисленное с шагом , а - значение интеграла, вычисленное с шагом .

|  |  |
| --- | --- |
|  | (9) |

Однако формула (9) дает очень неточное значение погрешности и требует вычисления интеграла дважды. Для достижения более точного вычисления погрешности можно воспользоваться формулой (10) [3].

|  |  |
| --- | --- |
|  | (10) |

В формуле (10) a и b – границы интегрирования, dx – шаг интегрирования, а – максимальный модуль второй производной функции на точках промежутка.

### **3.3 Описание ручного счета тестового примера предложенным методом**

Согласно варианту, .

Интеграл: .

Подынтегральная функция: .

Пусть шаг интегрирования: .

Построим таблицу точек для интегрирования (таблица 2).

Таблица 2 - Сетка точек для интегрирования методом трапеций

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | 0,4 | 0,6 | 0,8 | 1,0 | 1,2 | 1,4 |
| y | 0,761 | 0,754 | 0,745 | 0,733 | 0,719 | 0,704 |

Найдем площади трапеций по формуле (7):

Найдем сумму площадей трапеций по формуле (8):

Найдем вторую производную функции .

Найдем для каждого таблицы 2:

Найдем погрешность вычислений по формуле (10):

Мы не учитываем погрешность округления значений при промежуточных расчетах в случае ручного счета.

Ответ: .

### **3.4 Разработка алгоритмов решения задачи**

Алгоритм решения задачи представлен в виде блок-схемы на рисунке 17.

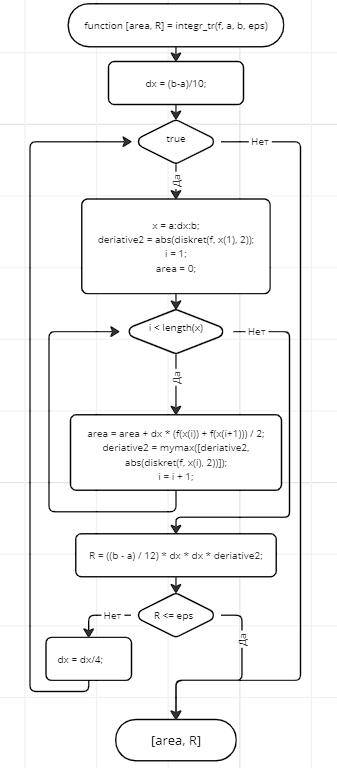


Рисунок 17 - Блок-схема функции “integr\_tr”

### **3.5 Листинг программы**

Реализация математического алгоритма с помощью MatLab состоит из нескольких функций: “integr\_tr”, “diskret”, “mymax”.

Функция “integr\_tr” является основной функцией, её код приведён в листинге 8.

*Листинг 8 - Функция “ingetr\_tr”*

|  |
| --- |
| function [area, R] = integr\_tr(f, a, b, eps)  % Функция, рассчитывающая значение определенного интеграла  % с учетом погрешности.  %  % f - подынтегральная функция  % a - нижняя граница интегрирования  % b - верхняя граница интегрирования  % eps - ограничение максимальной погрешности (значение погрешности  % вычислений не может превышатьмаксимальной погрешности, т.е. R <= eps)  %  % Результат - значение интеграла и погрешность вычислений    dx = (b-a)/10; % Задание шага интегрирования  while (true) % Значение интеграла будет расчитываться, пока не будет достигнута достаточная точность интегрирования  x = a:dx:b; % Построение узлов решетки на отрезке с заданным шагом  deriative2 = abs(diskret(f, x(1), 2)); % Поиск второй производной для последующего расчета погрешности  i = 1;  area = 0; % Инициализация переменной, хранящей сумму площадей трапеций  while (i < length(x))  area = area + dx \* (f(x(i)) + f(x(i+1))) / 2; % Расчет площади трапеции и добавление этого значения в переменную 'area'  deriative2 = mymax([deriative2, abs(diskret(f, x(i), 2))]); % Нахождение наибольшей второй производной  i = i + 1;  end  R = ((b - a) / 12) \* dx \* dx \* deriative2; % Расчет погрешности вычислений интеграла  if (R <= eps) % Проверка достаточной точности  break;  end  dx = dx/4; % Уменьшение шага интегрирования, согласно правилу Рунге  end  end |

Функция “diskret” описана в пункте 1.5, её код приведён в листинге 2.

Функция “mymax” находит максимальный элемент вектора значений, её код приведён в листинге 9.

*Листинг 9 - Функция “mymax”*

|  |
| --- |
| function answ = mymax(arr)  % Функция, реализующая поиск максимального элемента вектора.  %  % arr - вектор чисел  %  % Результат - число  if length(arr) >= 1  answ = arr(1); % Взятие в качестве начального значения первого элемента вектора  else  answ = 0;  end  i = 2;  while i <= length(arr)  if arr(i) > answ % Если очередной элемент больше найденного, найденный заменяется на него  answ = arr(i);  end  i = i + 1;  end |

### **3.6 Результат работы программы**

На рисунке 18 изображено подменю задания 3 программы.

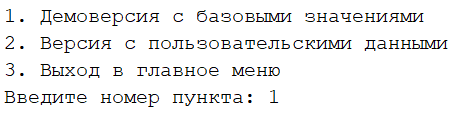


Рисунок 18 - Подменю 3, выбор версии решения

На рисунке 19 изображен результат работы демонстрационного варианта третьего задания.

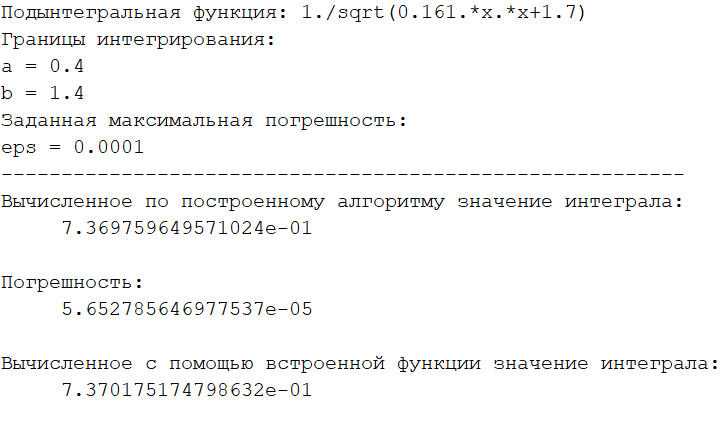
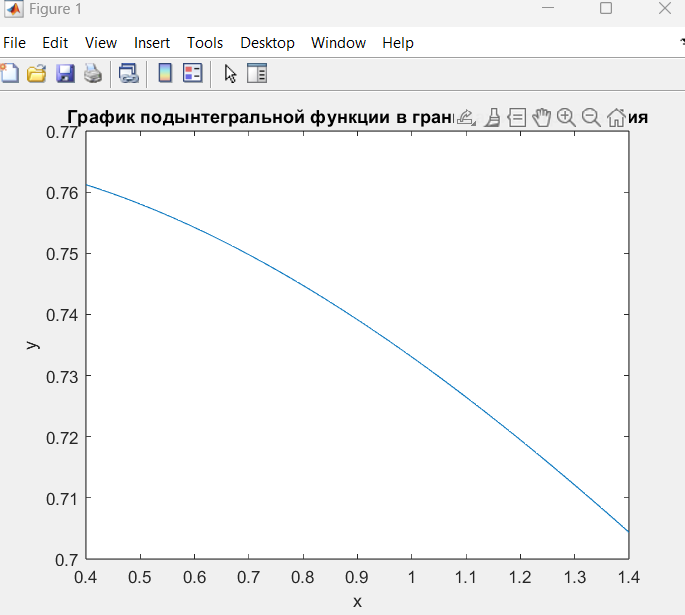


Рисунок 19 - Результат работы демоварианта 3 раздела

На рисунке 20 изображен график подынтегральной функции демоварианта.



*Рисунок 20 – График подынтегральной функции демоварианта*

На рисунке 21 изображен результат работы варианта с пользовательскими данными третьего задания.

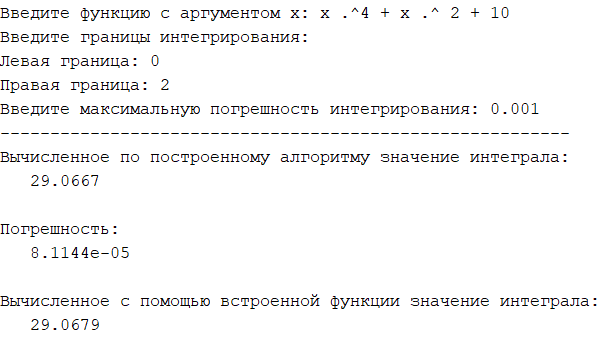
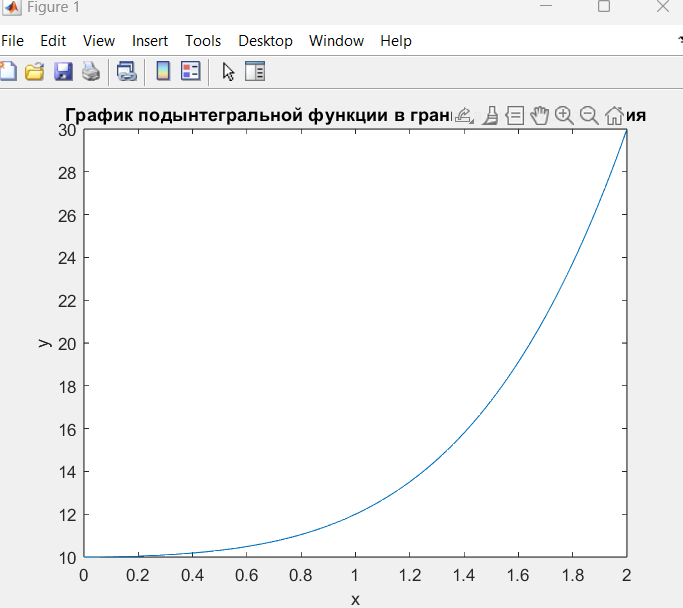


Рисунок 21 - Результат работы варианта с пользовательскими данными 3 раздела

На рисунке 22 изображен график подынтегральной функции варианта с пользовательскими данными.



*Рисунок 22 – График подынтегральной функции варианта с пользовательскими данными*

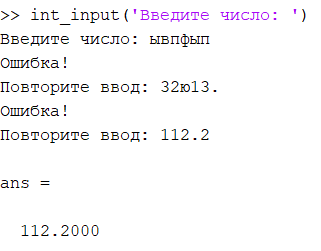
### **3.7 Результат реализации метода с помощью встроенных функций**

Для решения поставленной задачи, учитывая использование метода трапеций, должна использоваться встроенная функция “trapz”. Результат работы продемонстрирован на рисунках 19 (демовариант) и 20 (пользовательские значения).

## **4 Раздел 4. Обработка ошибок**

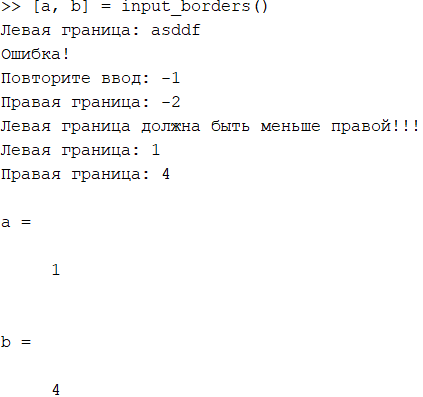
Во время работы программы возможно возникновение ошибок в связи с тем, что пользователь может ввести некорректные данные на вход программы.

Обработка ввода нечислового значения при вводе числа представлена на рисунке 23.



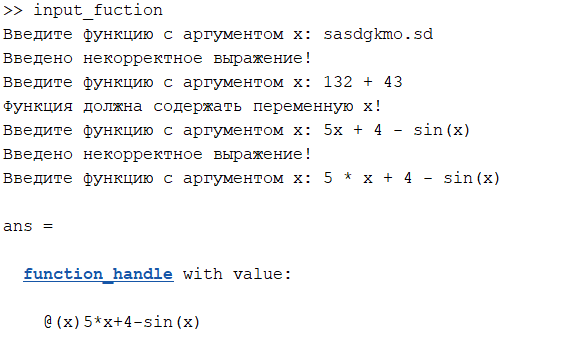
*Рисунок 23 – Обработка ошибки ввода числа*

Обработка неправильного ввода границ какого-либо промежутка представлена на рисунке 24.



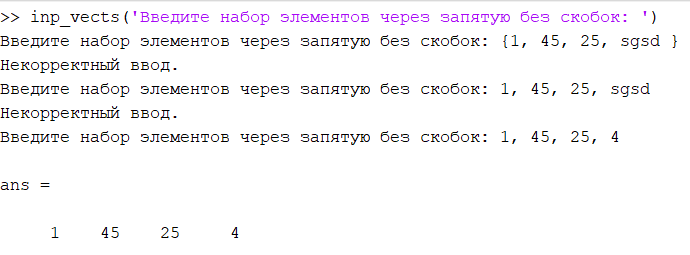
*Рисунок 24 – Обработка ошибки ввода границ промежутка*

Обработка неправильного ввода пользовательской функции представления на рисунке 25.



*Рисунок 25 – Обработка ошибки ввода пользовательской функции*

Обработка неправильного ввода набора значений для получения вектора представлена на рисунке 26.



*Рисунок 26 – Обработка ошибки ввода вектора элементов*

# **ЗАКЛЮЧЕНИЕ**

## **Решение нелинейных и трансцендентных уравнений**

В процессе выполнения раздела 1 изучен метод Ньютона (касательных) для решения нелинейных уравнений. Разработана проверка функции на возможность применения метода Ньютона. Получен опыт нахождения первой и второй производной средствами программирования. Ручной счет совпал с компьютерным в пределах погрешности ().

Корректность реализации математического метода подтверждается совпадением в пределах погрешности значений, полученных также с помощью встроенной функции “fzero”.

## **Интерполяция**

В процессе выполнения раздела 2 изучен метод интерполяции полиномом Лагранжа. Получен опыт работы с полиномами в представлении векторов коэффициентов для программной реализации. Ручной счет совпал с компьютерным с незначительной погрешностью, возникшей при округлении в ручном счете ().

Корректность реализации математического метода подтверждается совпадением в пределах погрешности значений, полученных с помощью встроенных функций “polyfit” и “polyval”.

## **Численное интегрирование**

В процессе выполнения раздела 3 изучен метод трапеций для интегрирования. Получен опыт оценки погрешности численного вычисления интеграла методом трапеций. Ручной счет совпал с компьютерным с незначительной погрешностью, возникшей при округлении в ручном счете ().

Корректность реализации математического метода подтверждается совпадением в пределах погрешности значений, полученных также с помощью встроенной функции “trapz”.

Таким образом, выполнив задание на учебную практику, закрепил знания языка программирования Matlab и умение вести с его помощью разработку комплексных программ, ознакомился с тремя математическими методами и реализовал их с помощью MatLab в программе, создал текстовый интерфейс для взаимодействия пользователя с программой.

Задание выполнял поэтапно, разделив работу на выполнение каждого раздела соответственно. Каждый этап состоял, в свою очередь, из следующих шагов:

1. Изучение численного математического метода для той или иной задачи, выделение шагов алгоритма, необходимых формул, ограничений на исходные данные;
2. Ручной счёт по изученному в первом пункте алгоритму;
3. Перенос того же алгоритма в вид блок-схемы для последующей реализации в коде программы;
4. Реализация алгоритма в программе на языке MatLab, а также поиск встроенной функции MatLab, аналогичной реализуемой, и решение задачи с её помощью, вывод обоих результатов вместе для сравнения. Код всех скриптов и функций программы обладает подродными комментариями, помогающими разобраться в архитектуре программы.

После завершения работы над каждым из трёх этапов, разработал текстовый интерфейс программы, из которого пользователь может выбрать раздел и вариант решения (демовариант или с пользовательскими данными), объединил три раздела в одной программе, включил в главное меню интерфейса пункт “Информация о программе и авторе”, где оставил необходимую информацию и свои контанты для связи.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Авхадиев Ф.Г. Учебно-методическое пособие по численным методам анализа: учебно-методическое пособие / Ф. Г. Авхадиев, Р. К. Губайдуллина, Р. Г. Насибуллин – Казань: Казанский (Приволжский) федеральный университет, 2019. – 113 с.
2. Ключарев А. А. Информатика. Алгоритмизация и структурное программирование в MATLAB: учебное пособие / А. А. Ключарев, А. А. Фоменкова, А. В. Туманова – СПб: ГУАП, 2019. - 143 с.
3. Кочегуров А.И. Теория и реализация задач вычислительной математики в пакете MathCad: учебное пособие / А. И. Кочегуров, Е. А. Кочегурова – Томск: Томский политехнический университет, 2013. – 135 с.

# **ПРИЛОЖЕНИЯ**

*Приложение А.1 – Функция “coefs2func”*

|  |
| --- |
| function f = coefs2func(coefs)  % Функция, формирующая Function Handle из вектора коэффициентов полинома.  %  % Коэффициенты распологаются в порядке от коэфф. младшего разряда (слева)  % до старшего (справа).  %  % Результат - функция, т.е. Function Handle  s = ''; % Создание строки, в которую будет формироваться функция из вектора коэффициентов  i = 1;  while (i <= length(coefs)) % Обработка коэффициентов  if (coefs(i) > 0)  if (coefs(i) == 1)  s = strcat('+x.^', num2str(i-1, 10), s);  else  s = strcat('+', num2str(coefs(i), 10), '\*x.^', num2str(i-1, 10), s);  end  else  if (coefs(i) < 0)  if (coefs(i) == -1)  s = strcat('-x.^', num2str(i-1, 10), s);  else  s = strcat(num2str(coefs(i), 10), '\*x.^', num2str(i-1, 10), s);  end  end  end  i = i + 1;  end  f = eval(strcat('@(x)', s)); % Преобразование полученной строки в Function Handle  end |

*Приложение А.2 – Скрипт “demo\_razdel\_1”*

|  |
| --- |
| close all;  disp('f = 5\*exp(-3\*x).\*sin(1\*x+0.7)-1'); % Вывод вида демо-функции для удобства пользователя  f = @(x)5\*exp(-3\*x).\*sin(1\*x+0.7)-1; % Задание демо-функции  disp('--------------------------------');  % Границы построения графика  a\_graph = -2;  b\_graph = 2;    t = a\_graph:(b\_graph-a\_graph)/1000:b\_graph;  y = f(t);  plot(t, y, t, zeros(size(t))); % Строим график  title('График функции в указанных границах');  xlabel('x');  ylabel('y');  xlim([a\_graph-(b\_graph-a\_graph)/50, b\_graph+(b\_graph-a\_graph)/50]); % Ограничиваем график для удобства использования  if abs(mymax(y)-mymin(y)) < 40  ylim([mymin(y)-(mymax(y)-mymin(y))/20, mymax(y)+(mymax(y)-mymin(y))/20]);  else  ylim([-20, 20]);  end  drawnow;В  % Границы поиска корня  left\_t = 0;  right\_t = 1;  disp('Границы поиска корня:');  disp(left\_t);  disp(right\_t);  disp('--------------------------------');  disp('Погрешность: 0.001');  epsilon = 0.001; % Погрешность поиска  res = solve\_1\_Newton(f, left\_t, right\_t, epsilon); % Вызов функции, реализующей алгоритм ньютона  opt = optimset('TolX', epsilon); % Задание точности для встроенной функции  mlb\_res = fzero(f, [left\_t, right\_t], opt); % Вызов встроенной функции, близкой к алгоритму ньютона  disp('--------------------------------');  disp('Результат:');  disp(res); % Вывод результата  disp('Результат через встроенную функцию:');  disp(mlb\_res); % Вывод результата, полученного с помощью встроенной функции |

*Приложение А.3 – Скрипт “demo\_razdel\_2”*

|  |
| --- |
| close all;  x = [1.2, 1.5, 1.7, 2, 2.4, 2.6, 3.1, 3.3, 3.5, 3.9]; % Задание вектора X (начальные аргументы)  y = [7.277, 7.545, 7.811, 8.089, 8.357, 8.626, 8.893, 9.16, 9.433, 9.701]; % Задание вектора Y (начальные значения)  x0 = [1.25, 3.55]; % Задание вектора X0 (аргументы для искомых значений)  disp('Таблично введенная функция:');  disp('x | 1.2 | 1.5 | 1.7 | 2 | 2.4 | 2.6 | 3.1 | 3.3 | 3.5 | 3.9');  disp('---------------------------------------------------------------');  disp('y | 7.277 | 7.545 | 7.811 | 8.089 | 8.357 | 8.626 | 8.893 | 9.16 | 9.433 | 9.701');  disp('Аргументы искомых (интерполированных) значений:');  disp('x0 = [1.25, 3.55]');  disp('-----------------------');  f = lagrange(x, y); % Вызов разработанной функции, возвращающий функцию-полином, согласно методу Лагранжа  disp('Искомые значения (в виде точек) c использованием функции lagrange:');  show(f, x0); % Поиск и вывод в консоль значений для искомых точек  x1 = x(1):(x(length(x))-x(1))/100:x(length(x)); % Создание вектора аргументов для построения графика функции  y1 = f(x1); % Создание вектора значений для построения графика функции  y0 = f(x0); % Создание вектора искомых значений для построения искомых точек на графике  % Построение графика для функции, вычисленной с помощью разработанной  % функции lagrange  hold on;  title('График функции, полученной с помощью разработанной функции lagrange');  plot(x, y, 'o','LineWidth', 2);  plot(x1, y1);  plot(x0, y0, '\*', 'MarkerSize', 10, 'LineWidth', 2');  xlabel('x');  ylabel('y');  legend({'Начальные точки','Полученная функция', 'Искомые точки'},'Location','southeast');  hold off;  % Расчет функции и значений с использованием встроенных функций  p = polyfit(x, y, length(x)-1);  y1 = polyval(p, x1);  y0 = polyval(p, x0);  disp('Искомые значения (в виде точек) c использованием встроенных функций:');  show(coefs2func(myflip(p)), x0); % Поиск и вывод в консоль значений для искомых точек  % Построение графика для функции, вычисленной с помощью встроенных функций  % polyfit и polyval  figure;  hold on;  title('График функции, полученной с помощью встроенных функций polyfit и polyval');  plot(x, y, 'o','LineWidth', 2);  plot(x1, y1);  plot(x0, y0, '\*', 'MarkerSize', 10, 'LineWidth', 2');  xlabel('x');  ylabel('y');  legend({'Начальные точки','Полученная функция', 'Искомые точки'},'Location','southeast');  hold off;  drawnow; |

*Приложение А.4 – Скрипт “demo\_razdel\_3”*

|  |
| --- |
| close all;  disp('Подынтегральная функция: 1./sqrt(0.161.\*x.\*x+1.7)');  f = @(x) 1 ./ sqrt(0.161 .\* x .\* x + 1.7); % Задание подынтегральной функции  disp('Границы интегрирования:');  disp('a = 0.4');  disp('b = 1.4');  a = 0.4; % Нижняя граница интегрирования  b = 1.4; % Верхняя граница интегрирования  disp('Заданная максимальная погрешность:');  disp('eps = 0.0001');  eps = 0.0001; % Погрешность вычисления интеграла  [area, R] = integr\_tr(f, a, b, eps); % Вычисление интеграла с учетом полученной погрешности R  disp('---------------------------------------------------------');  disp('Вычисленное по построенному алгоритму значение интеграла:')  disp(area); % Вывод результата  disp('Погрешность:')  disp(R); % Вывод погрешности  x = a:(b-a)/100:b;  y = f(x);  plot(x, y);  title('График подынтегральной функции в границах интегрирования');  xlabel('x');  ylabel('y');  drawnow;  area = trapz(x, f(x)); % Вычисление интеграла с помощью встроенной функции  disp('Вычисленное с помощью встроенной функции значение интеграла:')  disp(area); % Вывод результата |

*Приложение А.5 – Функция “diskret”*

|  |
| --- |
| function rez = diskret(f, x, counter)  % Функция, определяющая производную (первую или вторую,  % в зависимости от параметра counter) функции f в точке x.  %  % При counter == 1 - первая производная  % При counter ~= 1 - вторая производная  %  % Результат - числовое значение соответствующей производной  if counter == 1  rez = (f(x + 0.0001) - f(x))/0.0001;  else  rez = (f(x + 0.0002) - 2\*f(x + 0.0001) + f(x))/0.00000001;  end |

*Приложение А.6 – Скрипт “full\_razdel\_1”*

|  |
| --- |
| close all;  %f = '5\*exp(-3\*x).\*sin(1\*x+0.7)-1';  f = input\_fuction(); % Ввод пользовательской функции  disp('--------------------------------');  disp('Сейчас вам предстоит построить график, чтобы');  disp('затем по нему определить границы для поиска корня.');    while true % Отрисовка графика для пользователя  disp('Введите границы для отображения графика.');  [a\_graph, b\_graph] = input\_borders(); % Dводим границы для графика  t = a\_graph:(b\_graph-a\_graph)/1000:b\_graph;  y = f(t);  plot(t, y, t, zeros(size(t))); % Cтроим график  title('График функции в указанных границах');  xlabel('x');  ylabel('y');  xlim([a\_graph-(b\_graph-a\_graph)/50, b\_graph+(b\_graph-a\_graph)/50]); % Ограничиваем график для удобства использования  if abs(mymax(y)-mymin(y)) < 40  ylim([mymin(y)-(mymax(y)-mymin(y))/20, mymax(y)+(mymax(y)-mymin(y))/20]);  else  ylim([-20, 20]);  end  drawnow; % Принудительно запускаем график, находясь в цикле  disp('--------------------------------');  answer = input('Для продолжения введите любой символ.\nЧтобы перестроить график, введите R: ', 's');  if ~(all(answer == 'R') && length(answer) == 1) % По желанию пользователя продолжаем работу или перестраиваем график  break  end  end    disp('--------------------------------');  disp('Введите границы поиска корня.');  [left\_t, right\_t] = input\_borders(); % Ввод границ промежутка для поиска корня  b\_point = 'n';  while ~segment\_check(f, left\_t, right\_t) % Проверка промежутка на непрерывность, монотонность, одинаковую выпуклость  disp('Ошибочно выбран промежуток!');  disp('На промежутке функция должна быть:');  disp('-непрерывна;');  disp('-монотонна;');  disp('-не менять выпуклости.');  disp('-иметь разные знаки (для единичного пересечения оси Ox).');  disp('Введите границы поиска корня снова.');  b\_point = input('Если хотите прервать выполнение программы, введите q:', 's');  if b\_point == 'q'  break  end  [left\_t, right\_t] = input\_borders(); % Ввод границ промежутка для поиска корня  end  if b\_point == 'q' % Выход в меню при выборе пользователя  disp('Выполнение программы прервано');  else  disp('--------------------------------');  epsilon = inaccuracy('Введите погрешность поиска корня: '); % Ввод погрешности поиска  res = solve\_1\_Newton(f, left\_t, right\_t, epsilon); % Вызов функции, реализующей алгоритм ньютона  opt = optimset('TolX', epsilon); % Задание точности для встроенной функции  mlb\_res = fzero(f, [left\_t, right\_t], opt); % Вызов встроенной функции, близкой к алгоритму ньютона  disp('--------------------------------');  disp('Результат:');  disp(res); % Вывод результата  disp('Результат через встроенную функцию:');  disp(mlb\_res); % Вывод результата, полученного с помощью встроенной функции  disp('--------------------------------');  % Пока пользователь не захочет выйти, будет работать демовариант  end |

*Приложение А.7 – Скрипт “full\_razdel\_2”*

|  |
| --- |
| close all;  x = inp\_vects('Введите вектор значений X (ч/з запятую, без скобок):'); % Задание вектора X (начальные аргументы)  while (~myunique(x)) % Проверка на уникальность аргументов функции  disp('Вектор аргументов обязан быть уникален!');  x = inp\_vects('Введите вектор значений X (ч/з запятую, без скобок):'); % Задание вектора X (начальные аргументы)  end  y = inp\_vects('Введите вектор значений Y (ч/з запятую, без скобок):'); % Задание вектора Y (начальные значения)  while (length(x) ~= length(y))  disp('Количество элементов в векторах X и Y должно совпадать!'); % Проверка на правильность введения векторов аргументов и значений  y = inp\_vects('Введите вектор значений Y (ч/з запятую, без скобок):');  end  x0 = inp\_vects('Введите вектор значений Х0 (ч/з запятую, без скобок):'); % Задание вектора X0 (аргументы для искомых значений)  while (mymax(x0) > mymax(x) || mymin(x0) < mymin(x)) % Проверка на правильность введения вектора Х0  disp('При интерполяции значения должны искаться только для аргументов, находящихся мужду известными точками!');  x0 = inp\_vects('Введите вектор значений X0 (ч/з запятую, без скобок):');  end  f = lagrange(x, y); % Вызов разработанной функции, возвращающий функцию-полином, согласно методу Лагранжа  disp('Искомые значения (в виде точек) c использованием функции lagrange:');  show(f, x0); % Поиск и вывод в консоль значений для искомых точек  x1 = x(1):(x(length(x))-x(1))/100:x(length(x)); % Создание вектора аргументов для построения графика функции  y1 = f(x1); % Создание вектора значений для построения графика функции  y0 = f(x0); % Создание вектора искомых значений для построения искомых точек на графике  % Построение графика для функции, вычисленной с помощью разработанной  % функции lagrange  hold on;  title('График функции, полученной с помощью разработанной функции lagrange');  plot(x, y, 'o','LineWidth', 2);  plot(x1, y1);  plot(x0, y0, '\*', 'MarkerSize', 10, 'LineWidth', 2');  xlabel('x');  ylabel('y');  legend({'Начальные точки','Полученная функция', 'Искомые точки'},'Location','southeast');  hold off;  % Расчет функции и значений с использованием встроенных функций  p = polyfit(x, y, length(x)-1);  y1 = polyval(p, x1);  y0 = polyval(p, x0);  disp('Искомые значения (в виде точек) c использованием встроенных функций:');  show(coefs2func(myflip(p)), x0); % Поиск и вывод в консоль значений для искомых точек  % Построение графика для функции, вычисленной с помощью встроенных функций  % polyfit и polyval  figure;  hold on;  title('График функции, полученной с помощью встроенных функций polyfit и polyval');  plot(x, y, 'o','LineWidth', 2);  plot(x1, y1);  plot(x0, y0, '\*', 'MarkerSize', 10, 'LineWidth', 2');  xlabel('x');  ylabel('y');  legend({'Начальные точки','Полученная функция', 'Искомые точки'},'Location','southeast');  hold off;  drawnow; |

*Приложение А.8 – Скрипт “full\_razdel\_3”*

|  |
| --- |
| close all;  %f = @(x) 1 ./ sqrt(0.161 .\* x .\* x + 1.7);  f = input\_fuction(); % Ввод пользовательской подынтегральной функции  disp('Введите границы интегрирования:');  [a, b] = input\_borders(); % Ввод границ интегрирования  eps = inaccuracy('Введите максимальную погрешность интегрирования: '); % Ввод погрешности вычисления интеграла  [area, R] = integr\_tr(f, a, b, eps); % Вычисление интеграла с учетом полученной погрешности R  disp('---------------------------------------------------------');  disp('Вычисленное по построенному алгоритму значение интеграла:')  disp(area); % Вывод результата  disp('Погрешность:')  disp(R); % Вывод погрешности  x = a:(b-a)/100:b;  y = f(x);  plot(x, y);  title('График подынтегральной функции в границах интегрирования');  xlabel('x');  ylabel('y');  drawnow;  area = trapz(x, f(x)); % Вычисление интеграла с помощью встроенной функции  disp('Вычисленное с помощью встроенной функции значение интеграла:')  disp(area); % Вывод результата |

*Приложение А.9 – Функция “inacuracy”*

|  |
| --- |
| function output = inaccuracy(content)  % Функция для ввода погрешности.  %  % Результат - числовое значение погрешности  output = int\_input(content);  while output <= 0 % Погрешность - строго положительное значение  disp('Погрешность должна быть строго положительным значением!');  output = int\_input(content);  end  end |

*Приложение А.10 – Скрипт “info\_prog”*

|  |
| --- |
| disp('Программа ''MyLib''');  disp('Назначение: автоматизация математических задач');  disp('(Решение нелинейных уравнений, интерполяция, интегрирование)');  disp('- - - - - - - - - - - - - - -');  disp('В программе используются следующие методы решений:');  disp('Нелинейные уравнения - метод Ньютона');  disp('Интерполяция - метод Лагранжа');  disp('Интегрирование - метод трапеций');  disp('- - - - - - - - - - - - - - -');  disp('Последняя редакция: 21.04.2023');  disp('- - - - - - - - - - - - - - -');  disp('Разработчик:');  disp('Куриш Михаил Викторович');  disp('ГУАП, гр №4232');  disp('kurish04@mail.ru');  disp('- - - - - - - - - - - - - - -');  disp('');  esc = input('Для продолжения введите любой символ: ', 's'); |

*Приложение А.11 – Функция “inp\_vects”*

|  |
| --- |
| function outputArg = inp\_vects(content)  % Функция для ввода вектора значений с последующей конвертацией в вектор  % в представлении MatLab.  %  % Результат - вектор значений  while true  outputArg = input(content, 's'); % Ввод строки - набора значений ч/з запятую  try  outputArg = eval(['[', outputArg, ']']); % Попытка получить из строки вектор  if (isnumeric(outputArg)) % Если введен вектор не из чисел, то ввод выполнится заново  break;  end  catch  end  disp('Некорректный ввод.'); % Обработка ошибки  end |

*Приложение А.12 – Функция “input\_borders”*

|  |
| --- |
| function [arg1,arg2] = input\_borders()  % Функция для ввода границ отрезка.  %  % Результат - значения левой и правой границ отрезка соответственно  arg1 = int\_input('Левая граница: ');  arg2 = int\_input('Правая граница: ');  while arg1 >= arg2 % Проверка на правильность ввода границ  disp('Левая граница должна быть меньше правой!!!');  arg1 = int\_input('Левая граница: ');  arg2 = int\_input('Правая граница: ');  end  end |

*Приложение А.13 – Функция “input\_function”*

|  |
| --- |
| function output = input\_fuction()  % Функция для ввода пользовательской математической функции.  %  % Результат - анонимная функция пользовательской математической функции  while true  f = input('Введите функцию с аргументом x: ', 's'); % Ввод функции  f = strrep(f,'.\*','\*');  f = strrep(f,'./','/');  f = strrep(f,'.^','^');  f = strrep(f,'\*','.\*');  f = strrep(f,'/','./');  f = strrep(f,'^','.^');  flag = true;  while flag  try  output = eval(['@(x)', f]); % Формирование function handle из строки  a = output(2); % Проверка работоспособности функции  flag = false;  catch  disp('Введено некорректное выражение!');  f = input('Введите функцию с аргументом x: ', 's'); % Ввод функции  flag = true;  end  end  if mycontains(f, 'x') % Функция, в которой нет переменной, не может быть нелинейной  output = eval(['@(x)', f]); % Формирование function handle из строки  break;  else  disp('Функция должна содержать переменную x!');  end  end  end |

*Приложение А.14 – Функция “int\_input”*

|  |
| --- |
| function outputArg = int\_input(content)  % Функция для ввода численных значений с клавиатуры.  %  % Результат - числовое значение  outputArg = input(content, 's');  outputArg = replace(outputArg, ',', '.');  outputArg = str2double(outputArg);  while isnan(outputArg) % Проверка на правильность ввода  outputArg = input('Ошибка!\nПовторите ввод: ', 's');  outputArg = replace(outputArg, ',', '.');  outputArg = str2double(outputArg);  end  end |

*Приложение А.15 – Функция “integr\_tr”*

|  |
| --- |
| function [area, R] = integr\_tr(f, a, b, eps)  % Функция, рассчитывающая значение определенного интеграла  % с учетом погрешности.  %  % f - подынтегральная функция  % a - нижняя граница интегрирования  % b - верхняя граница интегрирования  % eps - ограничение максимальной погрешности (значение погрешности  % вычислений не может превышатьмаксимальной погрешности, т.е. R <= eps)  %  % Результат - значение интеграла и погрешность вычислений    dx = (b-a)/10; % Задание шага интегрирования  while (true) % Значение интеграла будет расчитываться, пока не будет достигнута достаточная точность интегрирования  x = a:dx:b; % Построение узлов решетки на отрезке с заданным шагом  deriative2 = abs(diskret(f, x(1), 2)); % Поиск второй производной для последующего расчета погрешности  i = 1;  area = 0; % Инициализация переменной, хранящей сумму площадей трапеций  while (i < length(x))  area = area + dx \* (f(x(i)) + f(x(i+1))) / 2; % Расчет площади трапеции и добавление этого значения в переменную 'area'  deriative2 = mymax([deriative2, abs(diskret(f, x(i), 2))]); % Нахождение наибольшей второй производной  i = i + 1;  end  R = ((b - a) / 12) \* dx \* dx \* deriative2; % Расчет погрешности вычислений интеграла  if (R <= eps) % Проверка достаточной точности  break;  end  dx = dx/4; % Уменьшение шага интегрирования, согласно правилу Рунге  end  end |

*Приложение А.16 – Функция “lagrange”*

|  |
| --- |
| function f = lagrange(x, y)  % Функция, реализующая построение функции по  % методу интерполяционного полинома Лагранжа.  %  % x - вектор аргументов  % y - вектор значений для соответствующих аргументов  %  % Результат - function handle, функция  i = 1;  coefs = []; % Создание вектора коэффициентов полинома  while (i <= length(y))  coefs = sloj(coefs, mnoj(lix(x,i), y(i))); % Суммирование произведений значений (y(i)) с соответствующими произведениями дробей (lix(x,i))  i = i + 1;  end  f = coefs2func(coefs); % Преобразование вектора коэффициентов в function handle  end |

*Приложение А.17 – Функция “lix”*

|  |
| --- |
| function coefs = lix(x, i)  % Функция, реализующая произведения всех дробей вида (x-xj)/(xi-xj)  % по вектору x для определенного i соответственно  % методу интерполяционного полинома Лагранжа.  %  % Вычисления происходят с векторами коэффициетов полиномов.  % Конечный вектор - есть вектор коэффициентов полинома, являющегося  % произведением всех дробей, описанных выше.  %  % x - вектор аргументов  % i - значение индекса i в формуле произведений дробей по Лагранжу  %  % Результат - вектор коэффициентов  j = 1;  coefs = 1; % Создание начального вектора коэффициентов  while (j <= length(x))  if (j ~= i)  coefs = mnoj(coefs, [-x(j)/(x(i)-x(j)), 1/(x(i)-x(j))]); % Умножение вектора коэффициентов (полинома) на очередную дробь по формуле Лагранжа  end  j = j + 1;  end  end |

*Приложение А.18 – Функция “mnoj”*

|  |
| --- |
| function plnm = mnoj(lft, rght)  % Функция, реализующая умножение полиномов, представленных в виде  % векторов коэффициентов (разряды келичиваются слева-направо).  %  % lft - вектор коэфф. первого полинома  % rght - вектор коэфф. второго полинома  %  % Результат - вектор коэффициентов полинома  plnm = zeros(1, length(lft)+length(rght)-1); % Создание вектора коэффициентов итогового полинома  i = 1;  while (i <= length(rght))  j = 1;  while (j <= length(lft))  plnm(i+j-1) = plnm(i+j-1) + lft(j) \* rght(i); % Заполнение итогового вектора коэффициентов согласно правилам математики  j = j + 1;  end  i = i + 1;  end  end |

*Приложение А.19 – Функция “mycontains”*

|  |
| --- |
| function flg = mycontains(obj, sub)  % Функция, проверяющая содержание sub в obj.  %  % obj - вектор или строка  % sub - элемент вектора или символ строки соответственно  %  % Результат - логическое значение  i = 1;  flg = false; % Задание начального значения (флага)  while i <= length(obj)  if obj(i) == sub % Если значение sub содержится в obj, флаг меняется  flg = true;  break; % Проверка закончена  end  i = i + 1;  end  end |

*Приложение А.20 – Функция “myflip”*

|  |
| --- |
| function new\_arr = myflip(arr)  % Функция, инвертирующая вектор (право и лево меняются местами).  %  % arr - вектор  %  % Результат - вектор  l = length(arr);  new\_arr = zeros(1, l); % Создается новый вектор  i = 1;  while i <= l  new\_arr(l-(i-1)) = arr(i); % Новый вектор заполняется зеркально первому  i = i + 1;  end  end |

*Приложение А.21 – Функция “mymax”*

|  |
| --- |
| function answ = mymax(arr)  % Функция, реализующая поиск максимального элемента вектора.  %  % arr - вектор чисел  %  % Результат - число  if length(arr) >= 1  answ = arr(1); % Взятие в качестве начального значения первого элемента вектора  else  answ = 0;  end  i = 2;  while i <= length(arr)  if arr(i) > answ % Если очередной элемент больше найденного, найденный заменяется на него  answ = arr(i);  end  i = i + 1;  end |

*Приложение А.22 – Функция “mymin”*

|  |
| --- |
| function answ = mymin(arr)  % Функция, реализующая поиск минимального элемента вектора.  %  % arr - вектор чисел  %  % Результат - число  if length(arr) >= 1  answ = arr(1); % Взятие в качестве начального значения первого элемента вектора  else  answ = 0;  end  i = 2;  while i <= length(arr)  if arr(i) < answ % Если очередной элемент меньше найденного, найденный заменяется на него  answ = arr(i);  end  i = i + 1;  end |

*Приложение А.23 – Функция “myunique”*

|  |
| --- |
| function flg = myunique(arr)  % Функция, проверяющая вектор или строку на уникальность.  %  % arr - вектор или строка  %  % Результат - логическое значение  second = zeros(length(arr)); % Создание вспомогательного вектора  second(1) = arr(1); % Первый элемент сам по себе уникален в сравнении с пустым множеством  i = 2; % Начиная со второго элемента будет выполняться проверка уникальности  flg = true; % Задаем начальное значение (флаг)  while i <= length(arr)  if mycontains(second(1:i-1), arr(i)) % Если во вспомогательном векторе уже есть итерируемый элемент, то он не уникален  flg = false; % Меняем флаг, т.к. вектор не уникален  break; % Завершаем выполнение функции  end  second(i) = arr(i); % Добавляем элемент в вспомогательный вектор  i = i + 1;  end  end |

*Приложение А.24 – Скрипт “Project”*

|  |
| --- |
| % Отрисовка текстового меню  while true  % Предварительная очистка консоли и рабочей памяти  clc;  clear;  disp('1. Задание 1. Нелинейные и трансцендентные уравнения');  disp('2. Задание 2. Интерполяция и экстраполяция');  disp('3. Задание 3. Численное интегрирование');  disp('4. Информация о программе и авторе');  disp('5. Выход')  menu = input('Введите номер пункта: ', 's');  switch menu % При вводе с клавиатуры проверяется использование меню  case '1' % Первый раздел  clc;  clear;  while true % Отрисовка подменю 1 раздела  disp('1. Демоверсия с базовыми значениями');  disp('2. Версия с пользовательскими данными');  disp('3. Выход в главное меню');  sub\_menu = input('Введите номер пункта: ', 's');  switch sub\_menu % При вводе с клавиатуры проверяется использование подменю  case '1' % Демовариант  clc;  clear;  close all;  demo\_razdel\_1; % Вызов скрипта с кодом решения  case '2' % Вариант с пользовательской функцией  clc;  clear;  full\_razdel\_1; % Вызов скрипта с кодом решения  case '3'  clc;  clear;  break  otherwise % При ошибочном вводе  clc;  clear;  disp('Неверное значение, выберите пункт меню еще раз:')  end  end  case '2'  clc;  clear;  while true % Отрисовка подменю 2 раздела  disp('1. Демоверсия с базовыми значениями');  disp('2. Версия с пользовательскими данными');  disp('3. Выход в главное меню');  sub\_menu = input('Введите номер пункта: ', 's');  switch sub\_menu % При вводе с клавиатуры проверяется использование подменю  case '1' % Демовариант  clc;  clear;  close all;  demo\_razdel\_2; % Вызов скрипта с кодом решения  case '2' % Вариант с пользовательской функцией  clc;  clear;  full\_razdel\_2; % Вызов скрипта с кодом решения  case '3'  clc;  clear;  break  otherwise % При ошибочном вводе  clc;  clear;  disp('Неверное значение, выберите пункт меню еще раз:')  end  end  case '3'  clc;  clear;  while true % Отрисовка подменю 3 раздела  disp('1. Демоверсия с базовыми значениями');  disp('2. Версия с пользовательскими данными');  disp('3. Выход в главное меню');  sub\_menu = input('Введите номер пункта: ', 's');  switch sub\_menu % При вводе с клавиатуры проверяется использование подменю  case '1' % Демовариант  clc;  clear;  close all;  demo\_razdel\_3; % Вызов скрипта с кодом решения  case '2' % Вариант с пользовательской функцией  clc;  clear;  full\_razdel\_3; % Вызов скрипта с кодом решения  case '3'  clc;  clear;  break  otherwise % При ошибочном вводе  clc;  clear;  disp('Неверное значение, выберите пункт меню еще раз:')  end  end  case '4' % Вывод информации о программе и авторе  clc;  clear;  info\_prog;  case '5'  clc;  clear;  close all;  break  otherwise % При ошибочном вводе  clc;  clear;  disp('Неверное значение, выберите пункт меню еще раз:');  end  end |

*Приложение А.25 – Функция “segment\_check”*

|  |
| --- |
| function answ = segment\_check(f, a, b)  % Функция, проверяющая математическу функцию f на промежутке [a, b]  % на разные знаки в границах (для единичного пересечения оси Ox)  %  % Результат - логическое значение (true или false)  answ = true; % Задается стандартное значение для вывода  if f(a)\*f(b) >= 0 % Проверка значений в границах на разные знаки  answ = false;  return  end  end |

*Приложение А.26 – Функция “show”*

|  |
| --- |
| function show(f, x0)  % Функция, реализующая поиск и вывод в консоль значений для аргументов  % вектора x0 по заданной функции f.  % Вид:  % 'точка №i: (xi; yi)'  %  % f - function handle  % x0 - вектор аргументов  %  % Результат - (функция не имеет выходных значений)  i = 1;  while (i <= length(x0))  disp('точка №' + string(i) + ': (' + string(x0(i)) + '; ' + string(f(x0(i))) + ')');  i = i + 1;  end  end |

*Приложение А.27 – Функция “sloj”*

|  |
| --- |
| function plnm = sloj(lft, rght)  % Функция, реализующая сложение полиномов, представленных в виде  % векторов коэффициентов (разряды келичиваются слева-направо).  %  % lft - вектор коэфф. первого полинома  % rght - вектор коэфф. второго полинома  %  % Результат - вектор коэффициентов полинома  plnm = zeros(1, mymax([length(lft), length(rght)])); % Создание вектора коэффициентов итогового полинома  i = 1;  while (i <= length(lft))  plnm(i) = plnm(i) + lft(i); % Внесение в итоговый вектор значений первого вектора  i = i + 1;  end  i = 1;  while (i <= length(rght))  plnm(i) = plnm(i) + rght(i); % Сложение с имеющимися значениями итогового вектора значений второго вектора  i = i + 1;  end  end |

*Приложение А.28 – Функция “solve\_1\_Newton”*

|  |
| --- |
| function output = solve\_1\_Newton(f, a, b, e)  % Функция, реализующая поиск корня на интервале функции по  % методу Ньютона.  % f - function handle, функция  % a, b - левая и правая границы промежутка  % e - погрешность вычисления  %  % Результат - вычисленное значение корня или текст возникшей ошибки  d\_func = diskret(f, a, 1); % Определение производной в точке а  dd\_func = diskret(f, a, 2); % Определение второй производной  if dd\_func\*f(a) > 0 % Условие для точки а  output = a;  else  output = b;  end  format longE;  i = 0; % Счетчик количества итераций  divis = f(output)/d\_func; % Переменная, хранящее изменение функции в данной точке  accur = abs(divis); % Точность определения корня при данном значении корня  while accur > e % Пока точность недостаточна, выполнить алгоритм Ньютона  if i > 100  output = 'Слишком большое количество итераций!';  break;  end  output = output - divis; % Новый корень  divis = f(output)/diskret(f, output, 1);  accur = abs(divis); % Точность при новом корне  i = i + 1;  end  end |