[Введение 4](#_Toc27934084)

[1. Описание метода решения задачи 5](#_Toc27934085)

[1.1. Конечные разности 5](#_Toc27934086)

[1.2. Интерполяционная формула Ньютона 5](#_Toc27934087)

[2. Разработка структур данных 6](#_Toc27934088)

[3. Разработка алгоритмов 8](#_Toc27934089)

[4. Тестирование программы 13](#_Toc27934090)

[5. Разработка документации 20](#_Toc27934091)

[5.1. Назначение и условия применения программы 20](#_Toc27934092)

[5.2 Характеристика программы 20](#_Toc27934093)

[5.3 Обращение к программе 20](#_Toc27934094)

[5.4 Входные и выходные данные 24](#_Toc27934095)

[Заключение 25](#_Toc27934096)

[Библиографический список 26](#_Toc27934097)

[Приложение А. Листинг программы 27](#_Toc27934098)

Введение

Целью курсового проекта является разработка программной системы для интерполяции методом конечных разностей с помощью формулы Ньютона.

Задачами курсового проектирования является:

* изучение метода решения задачи;
* разработка алгоритма и структур данных;
* разработка программы, позволяющая решить задачу интерполяции методом конечных разностей;
* выполнить контрольный расчет в ручном режиме;
* провести тестирование работы программы на контрольном примере;
* выполнить тестирование работы программы на нескольких тестовых примерах;
* разработать документацию к программе.

Пояснительная записка к курсовому проекту состоит из следующих разделов:

* введение;
* описание метода решения задачи;
* разработка структур данных;
* разработка алгоритмов;
* тестирование программы;
* разработка документации;
* заключение;
* библиографический список;
* приложение (листинг программных модулей).

# Описание метода решения задачи

## Конечные разности

Конечной разностью первого порядка называется разность ∆yi = yi+1-yi, где yi+1 = f(xi+h) и yi = f(xi). Для функции, заданной таблично в (n+1) узлах, i = 0,1,2…n конечные разности первого порядка могут быть вычислены в точках 0, 1, 2, …, n-1:

∆y0 = y1-y0;   
∆y1 = y2-y1;  
………………….. ∆yn-1 = yn-yn-1.

Используя конечные разности первого порядка, можно получить конечные разности второго порядка ∆2yi = ∆yi+1-∆yi:

∆2y0 = ∆y1-∆y0;  
∆2y1 = ∆y2-∆y1;  
………………………∆2yn-2 = ∆yn-1-∆yn-2;

Конечные разности k-го порядка в узле с номером i могут быть вычислены через разности (k-1)-го порядка:

∆kyi = ∆k-1yi+1-∆k-1yi;

Любые конечные разности можно вычислить через значения функции в узлах интерполяции, например:

∆2y0 = ∆y1 - ∆y0 = (y2 - y­1) – (y1 - y0) = y2 - 2y1+ y0

## Интерполяционная формула Ньютона

Пусть y = f(x) – функция действительного переменного x. Для данного множества равностоящих значений аргумента xi = x0+i\*h где h – шаг таблицы, а i = 0,1, …, n, то для интерполяции могут применяться *формулы Ньютона*, использующие конечные разности.   
В курсовом проекте используется прямая интерполяционная формула Ньютона, которая выглядит следующим образом:

Где , yi = fi, а выражения вида – конечные разности.

# Разработка структур данных

Для решения задачи необходимо определить типы данных, которые будут использоваться для хранения исходных и промежуточных данных, а также результатов вычислений.

Для переменных в циклах необходимы данные целого типа. Язык *Free Pascal* предоставляет возможность использования следующих целочисленных типов (Таблица 1):

Таблица 1 – Целые типы данных

|  |  |  |
| --- | --- | --- |
| **Тип** | **Диапазон значений** | **Размер, байт** |
| Byte | 0 .. 255 | 1 |
| Shortint | -128 .. 127 | 1 |
| Smallint | -32768 .. 32767 | 2 |
| Word | 0 .. 65535 | 2 |
| Integer | smallint или longint | 2 или 4 |
| Cardinal | longword | 4 |
| Longint | -2147483648 .. 2147483647 | 4 |
| Longword | 0..4294967295 | 4 |
| Int64 | -9223372036854775808 .. 9223372036854775807 | 8 |
| QWord | 0 .. 18446744073709551615 | 8 |

Диапазон значений типов *Smallint, Word, Integer ,Cardinal*, *Longint*, *Longword*, *Int64*, *QWord* явно избыточен для решения поставленной задачи. В поставленной задаче не используются отрицательные значения, тип *Shortint* хоть и подходит по размеру, но имеет отрицательные значения. Поэтому будем использовать тип *Byte*.

Для реализации математических вычислений необходимы данные вещественного типа. Язык *Free Pascal* предоставляет возможность использования вещественных типов данных:

Таблица 2 – Вещественные типы данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон значений** | **Значимых разрядов** | **Размер, байт** |
| Real | Зависит от платформы | Зависит от платформы | 4 или 8 |
| Single | 1.5E-45 .. 3.4E38 | 7-8 | 4 |
| Double | 5.0E-324 .. 1.7E308 | 15-16 | 8 |
| Extended | 1.9E-4932 .. 1.1E4932 | 19-20 | 10 |
| Comp | -2E64+1 .. 2E63-1 | 19-20 | 8 |
| Currency | -922337203685477.5808 | 922337203685477.5807 | 8 |

Тип данных *Real* зависит от платформы, и использовать его нежелательно. Тип *Currency* предназначен для операций с числами с фиксированной точностью (операции с денежными величинами). Тип *Comp* трактуется как вещественное число без дробной части.

Точность данных типа *Single* недостаточна для решения задач вычислительной математики, поэтому предпочтение следует отдать типам *Double* или *Extended*.

# Разработка алгоритмов

Процесс нахождения значения y от заданной функции y=f(x) и записи результатов в массивы является итерационным процессом. Входными данными алгоритма будут:

* Начальное значение аргумента *x0*;
* Конечное значение аргумента *xi*;
* Шаг от начального значения к конечному *hx*;
* Максимальный размер массива *NMAX*;

Тело цикла будет включать в себя следующие действия:

* Присвоение значений *x* и *f(x)* элементам массива.

Условием завершения цикла является:

* Либо достижение конечного значения *xi;*
* Либо превышение заданного максимального значения массива *n >NMAX*;

Процесс нахождения конечных разностей включает в себя простые циклы с параметром. Используемыми данными будут являться значения функции y=f(x). Изобразим процесс нахождения конечных разностей:

Вывод   
y1,y2,y3

y1 = y[i+1]-y[i]  
y2 = y1[i+1] - y1[i]  
y3 = y2[i+1] - y2[i]

Цикл от 1 до n-k, k=1,2,3…

Процесс нахождения значения в указанной точке формулой Ньютона является итерационным процессом. Входными данными алгоритма будут:

* Начальное значение аргумента функции *x0;*
* Конечное значение аргумента функции *xi*;
* Шаг от начального к конечному значению *hx*;
* Полученная от пользователя точка *xk*;
* Максимальный размер массива *NMAX*;

Тело цикла будет включать в себя следующие действия:

* Определение принадлежности точки *xk*, введённой пользователем к промежутку от начального значения аргумента функции *x0* до конечного значения аргумента функции *xi*;
* Вложенный цикл, тело которого будет включать в себя:
  + - Вычисление ближайшего значения узла *xki* к введённому пользователем значению *xk*;

Условием завершения цикла будет:

* Превышение значения *xk* над значением узла *xki;*

Условием завершения цикла является:

* Принадлежность точки промежутку *x0* <= *xk* <= *xi*

На рисунке представлен алгоритм нахождения значения функции в заданной точке с помощью формулы Ньютона

# Тестирование программы

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом

Для тестирования программы интерполяции методом конечных разностей с помощью формулы Ньютона необходимо:

* Задать полином второй степени и найти его конечные разности;
* Ввести значение точки *x*, найти значение функции *f(x)* при помощи другого программного средства и проверить найденное решение с тем, что будет выдано программой.

Проведем тестирование программы для функции на промежутке от 1 до 8 с шагом 1.

Выполним тестовый запуск программы:

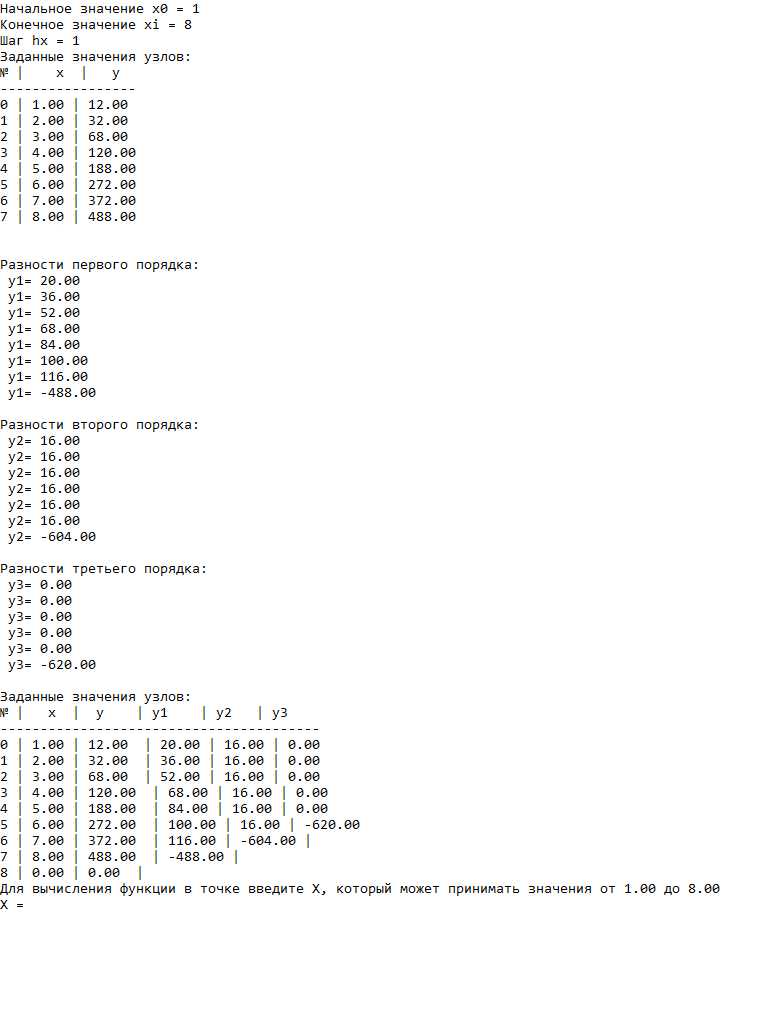


Рисунок 4.1 – Запуск программы

Найдём значение функции в точке, пусть это будет x = 3.45

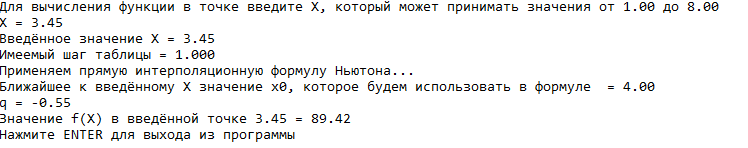


Рисунок 4.2 – Результаты выполнения программы

В результате запуска значение функции в указанной точке было найдено верно.

Проведём тестирование для полинома большей степени. на промежутке от 1 до 4 с шагом 0.2

Выполним тестовый запуск:

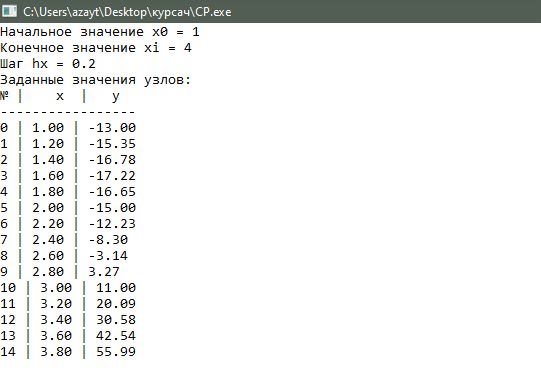


Рисунок 4.3 – Запуск программы

Найдём значение функции в точке, пусть это будет x = 3.45

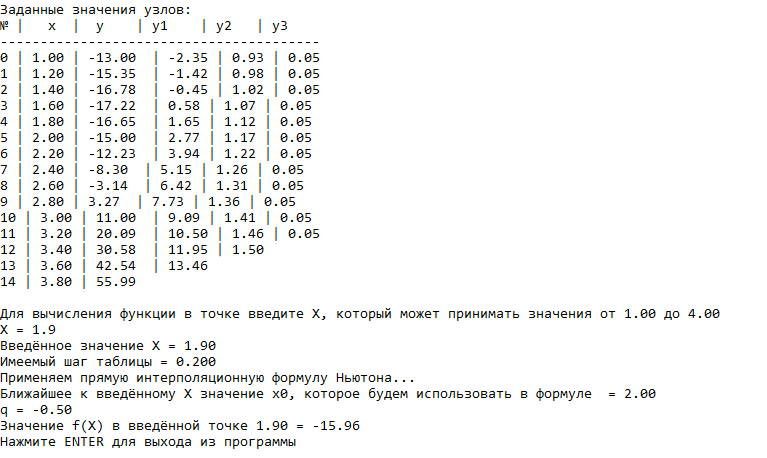


Рисунок 4.4 – Результаты выполнения программы

В результате запуска значение функции в указанной точке было найдено верно.

Проведём тестирование для функции на промежутке, превышающем допустимые значения, тем самым протестируем работу ограничений. Также построим график данной функции. Возьмём промежуток от 1 до 4 с шагом 0.1.

Выполним тестовый запуск:

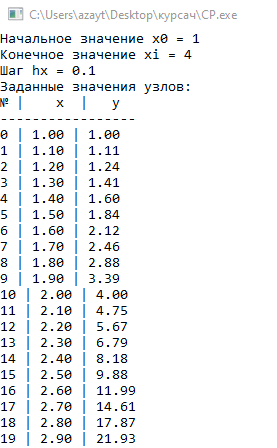


Рисунок 4.5 – Запуск Программы

Найдём значение функции в точке, пусть это будет x = 2.03

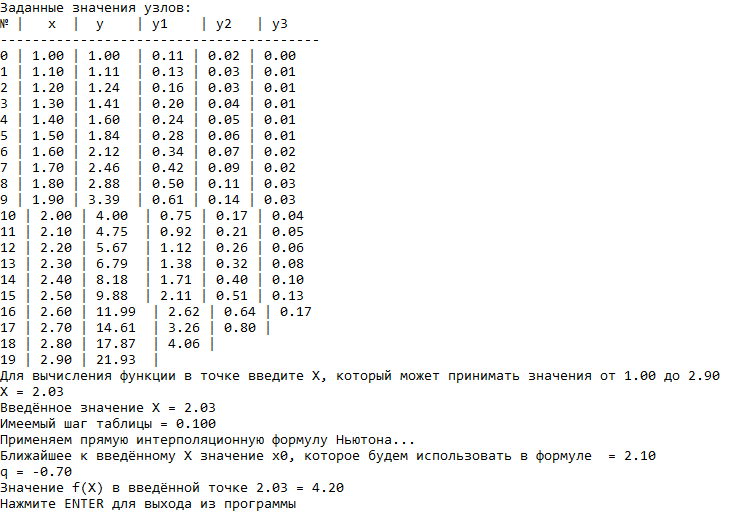


Рисунок 4.6 – Результаты выполнения программы

Алгоритм успешно остановил вычисление значения функции и конечных разностей по достижению заполнения максимального элемента массива, который равен 20.  
При этом, значение функции в заданной точке вычислено верно.   
Построим график.

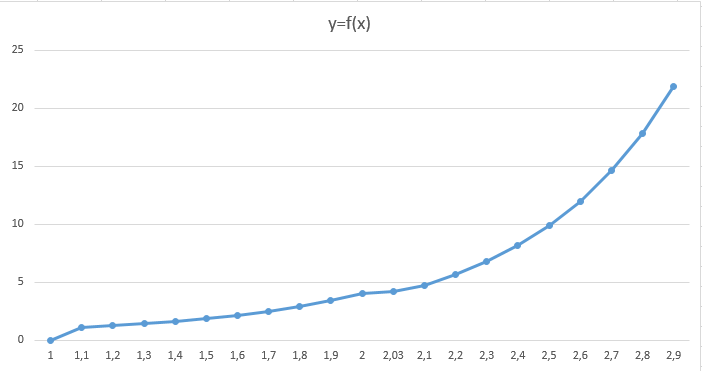


Рисунок 4.7 – график функции y=x^x

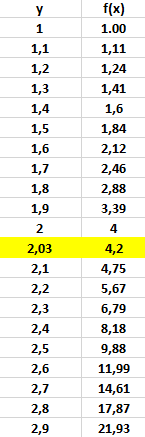


Рисунок 4.8 – таблица значений функции y=x^x

В таблице значений выделена искомая точка x.

По результатам испытаний считаем, что программа работает корректно.

# Разработка документации

В данном разделе приведено руководство программиста по использованию программы для интерполяции математической функции с помощью формулы Ньютона *Newton polynomial*.

## 5.1. Назначение и условия применения программы

Программа *Newton polynomial* предназначена для нахождения значения функции по формуле Ньютона с помощью конечных разностей. Программа позволяет задавать промежуток значений, в которых будет найдено значение функции, по ним составлены конечные разности и затем найдена заданная, заранее неизвестная точка *x*.

Для выполнения программы необходимо наличие операционной системы *Windows*. Особые требования к составу периферийных устройств не предъявляются.

Персонал, использующий программу (программист) должен обладать практическими навыками работы с текстовым пользовательским интерфейсом.

## 5.2 Характеристика программы

Программа *Newton polynomial* является независимым программным обеспечением и позволяет решить задачу вычислительной математики – интерполяция методом конечных разностей.

Работа программы осуществляется в консольном режиме.

Размер исполняемого файла *Newton Polynomial* – 129 059 байт.

## 5.3 Обращение к программе

Запуск программы осуществляется стандартным для операционной системы способом – двойным щелчком левой кнопки мыши на исполняемом файле *Newton\_Polynomial*.*exe*.

После запуска в окне консоли (Рисунок 5.1) появляется следующая информация:

* назначение программы;

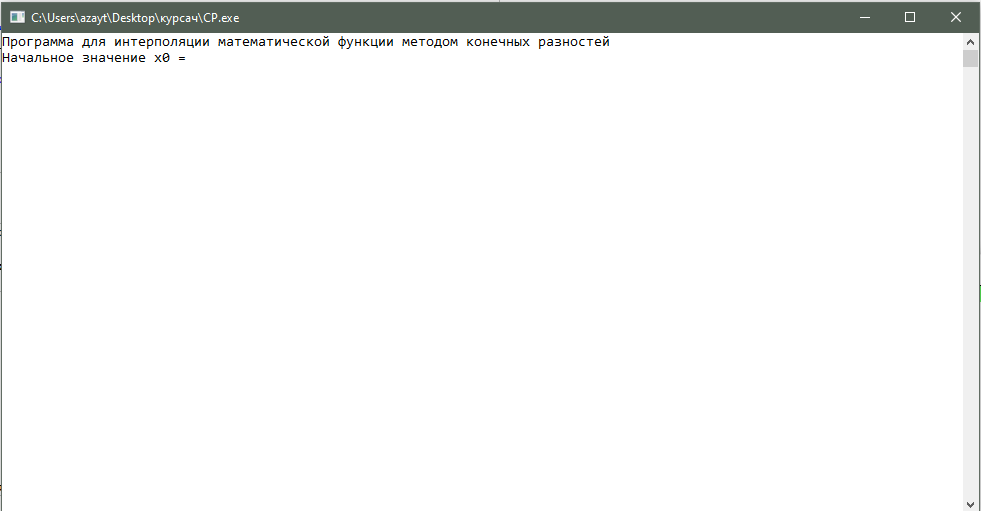


Рисунок 5.1 – Результат запуска программы Newton\_Polynomial

Пользователь должен ввести сначала начальное значение промежутка, затем конечное значение промежутка, затем шаг внутри этого промежутка, по которому будут строиться значения функции. (Рисунок 5.2)

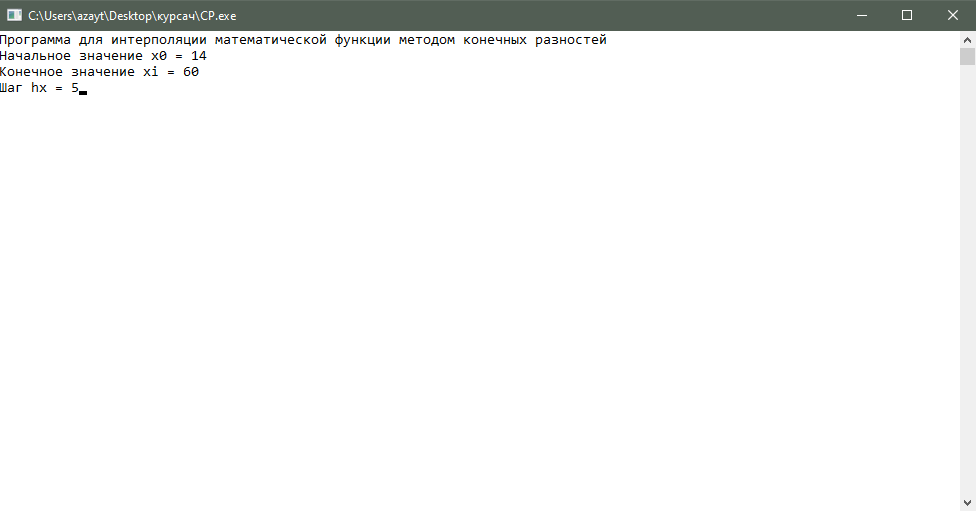


Рисунок 5.2 – Результат запуска программы Newton\_Polynomial

После ввода данных проверяется их корректность:

* левая граница должна быть меньше правой.

В случае ввода некорректных данных пользователю выводится сообщение об ошибке и запрашивается повторный ввод данных (Рисунок 5.3).

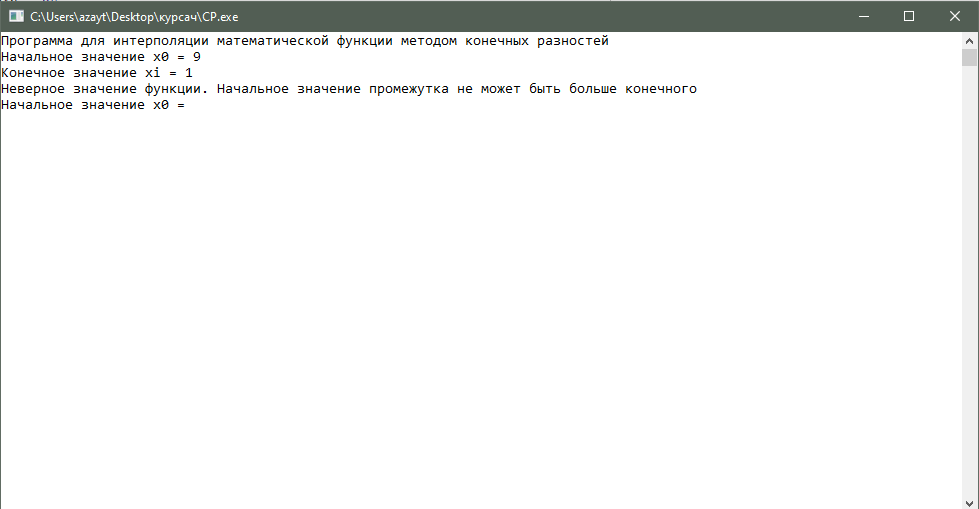


Рисунок 5.3 – Результат запуска программы Newton\_polynomial

Если введены корректные данные, то осуществляется расчет – значений функции в заданных точках и конечных разностей 1-го, 2-го и 3-го порядка. (Рисунок 5.4)

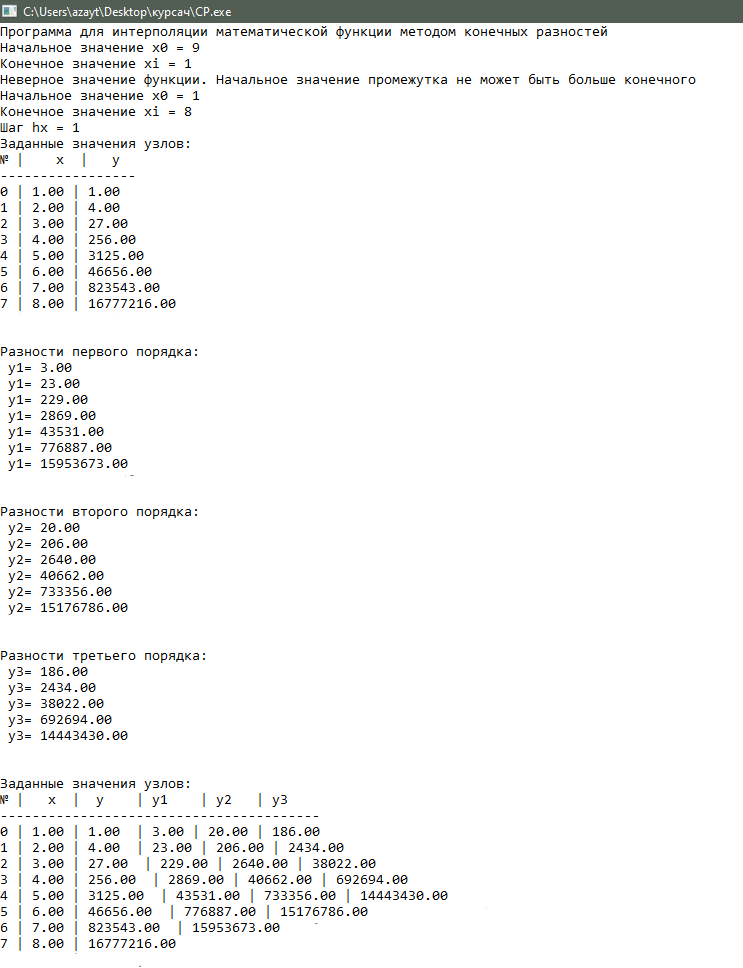


Рисунок 5.4 – Результат запуска программы Newton\_polynomial

## 5.4 Входные и выходные данные

Входными данными программы являются:

* исходное уравнение;
* Начальное и конечное значение промежутка;
* Шаг для аргумента *x*;
* Задаваемое пользователем значение аргумента *x*.

Исходное уравнение задается в виде функции. Начальное, конечное значение и шаг вводятся пользователем в процессе работы программы.

Выходными данными программы являются:

* Таблица значений узлов;
* Таблица конечных разностей 1-го, 2-го, 3-го порядков*;*
* Значение функции в заданной пользователем точке.

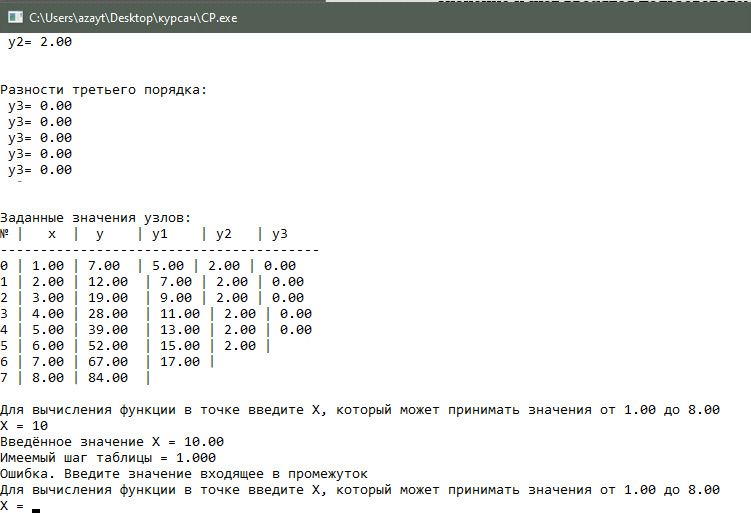


Рисунок 5.5 – Пример обработки некорректных данных

Заключение

В ходе данной курсовой работы решена задача интерполяции математический функций методом конечных разностей. Разработан алгоритм нахождения конечных разностей 1-го, 2-го, 3-го порядков. По этому алгоритму составлена и отлажена программа для вычисления значения функции в заданной точке с помощью прямой интерполяционной формулы Ньютона.

Библиографический список

1. ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов.
2. ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов.
3. ГОСТ 19.104-78\* ЕСПД. Основные надписи.
4. ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам.
5. ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом.
6. ГОСТ 19.504-79\* ЕСПД. Руководство программиста. Требования к содержанию и оформлению.
7. ГОСТ 19.604-78\* ЕСПД. Правила внесения изменения в программные документы, выполненные печатным способом.
8. Шарый С.П. Курс вычислительных методов. – Новосибирск, 2019.
9. Мудров А.Е. Численные методы для ПЭВМ на языках Бейсик, Фортран, Паскаль. – Томск: МП «Раско», 1991.
10. Корн Г., Корн Т. Справочник по математике – Москва, 1973.

Приложение А. Листинг программы

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

{ }

{ Проект Newton polynomial }

{ Copyright (c) 2019 ФГБОУ РГРТУ им. В.Ф. Уткина }

{ ФВТ/кафедра КТ }

{ }

{ Разработчик: ст. гр. 848 Зайцев А.Р. }

{ Модифицирован: 19 декабря 2019 }

{ }

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

program Newton\_polynomial;

//установка кодовой страницы для отображения русских символов

{$codepage utf8}

const

NMAX = 20; // Максимальный размер массива

type

TArr = array[1..NMAX] of double;

function f(x: double): double;

begin

f := x\*x+2\*x+4;

end;

var

x, y: TArr;

y1: TArr;

y2: TArr;

y3: TArr;

i, n, k: byte;

xi, x0, xn, hx, xki: double;

xk, q, otv: double;

begin

writeln('Программа для интерполяции математической функции методом конечных разностей');

repeat

write('Начальное значение x0 = '); readln(x0);

write('Конечное значение xi = '); readln(xn);

if x0>xn then

writeln('Неверное значение функции. Начальное значение промежутка не может быть больше конечного')

until x0<xn;

write('Шаг hx = '); readln(hx);

n := 1;

xi := x0;

while xi <= xn do

begin

x[n] := xi;

y[n] := f(xi);

n := n + 1;

xi := xi + hx;

if n > NMAX then // Защита от переполнения массива

begin

n := n - 1;

break;

end;

end;

writeln ('Заданные значения узлов:');

writeln ('№ | x | y ');

writeln ('-----------------');

for i:=1 to n do

writeln (i-1,' | ', x[i]:3:2,' | ', y[i]:3:2);

writeln();

writeln ('Разности первого порядка: ');

for i:=1 to n-1 do

begin

y1[i]:=y[i+1]-y[i];

writeln(' y1= ', y1[i]:4:2);

end;

writeln;

writeln ('Разности второго порядка: ');

for i:=1 to n-2 do

begin

y2[i]:=y1[i+1]-y1[i];

writeln(' y2= ',y2[i]:4:2);

end;

writeln;

writeln ('Разности третьего порядка: ');

for i:=1 to n-3 do

begin

y3[i]:=y2[i+1]-y2[i];

writeln(' y3= ',y3[i]:4:2);

end;

writeln;

writeln ('Заданные значения узлов:');

writeln ('№ | x | y | y1 | y2 | y3 ');

writeln ('----------------------------------------');

for i := 1 to n do

begin

write(i-1,' | ', x[i]:2:2,' | ', y[i]:2:2, ' | ');

if i < n then

write(y1[i]:2:2, ' | ');

if i < n - 1 then

write(y2[i]:2:2, ' | ');

if i < n - 2 then

write(y3[i]:2:2);

writeln;

end;

k:= n;

repeat

if n>NMAX then

writeln('Для вычисления функции в точке введите X, который может принимать значения от ', x0:2:2,' до ', x[i]:2:2)

else

writeln('Для вычисления функции в точке введите X, который может принимать значения от ', x0:2:2, ' до ', xi-1:2:2);

write('X = ');

read(xk);

writeln('Введённое значение X = ', xk:2:2);

writeln('Имеемый шаг таблицы = ', hx:3:3);

if (x0<=xk) and (xk<=xi) then

begin

writeln('Применяем прямую интерполяционную формулу Ньютона...');

for i:=1 to k-1 do

begin

if (xk>x[i]) then

k := k+1

else

begin

xki := x[i];

break;

end;

end;

writeln('Ближайшее к введённому X значение x0, которое будем использовать в формуле = ', xki:2:2);

q:=((xk-xki)/hx);

writeln('q = ', q:2:2);

otv := y[i]+y1[i]\*q+y2[i]\*(q\*(q-1))/2+y3[i]\*(q\*(q-1)\*(q-2))/6;

writeln('Значение f(X) в введённой точке ', xk:2:2,' = ', otv:2:2);

readln();

writeln('Нажмите ENTER для выхода из программы');

end

else

writeln('Ошибка. Введите значение, входящее в промежуток');

until (x0<=xk) AND (xk<=xi);

readln();

end.