|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | |
|  | | |  | | |  | | |
| ФГБОУ ВО «Пермский государственный  национальный исследовательский университет» | | | | | | | | |
|  | | |  | | |  | | |
|  | | ОТЧЕТ  по лабораторной работе № 5 «Интерполирование. Среднеквадратичное приближение» | | | | |  | |
|  | | |  | | |  | | |
|  | Работу выполнили  студенты гр. \_\_\_\_\_\_\_\_\_\_\_\_\_\_  Гришин Н.А.\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  Вотинова Е.С.\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 | | |  | Проверил  Профессор, доктор физико-математических наук  Русаков С.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 | | |  |
|  |  | | |  |  | | |  |
| Пермь 2020 | | | | | | | | |

Оглавление

[1. Постановка задачи 3](#_Toc55843261)

[2. Исходные данные 4](#_Toc55843262)

[3. Решение 5](#_Toc55843263)

[4. Тестирование 19](#_Toc55843264)

[5. Краткие выводы 24](#_Toc55843265)

[6. Текст программы. 25](#_Toc55843266)

1. Постановка задачи

*Задача* - приблизить заданную функцию  на отрезке .

1) Построить таблицу ,  .

По полученной таблице произвести интерполяцию с помощью

* + формулы Ньютона;
  + кубических сплайнов дефекта 1.

Провести оценку погрешности в узлах .

Сравнить оценку погрешности с реальной погрешностью.

2) Выполнить среднеквадратичное приближение заданной функции на заданном отрезке c помощью полинома второго порядка

* + дискретный вариант (по таблице из п.1);
  + непрерывный (интегральный) вариант.

Провести оценку погрешности.

Построить график приближаемой и приближающих функций.

3) Методом обратного интерполирования, используя интерполяционную формулу Ньютона, найти корень уравнения

.

Константа *с* выбирается таким образом, чтобы существовал корень на отрезке .

*Указание.*

1) При построении параметров кубического сплайна воспользоваться алгоритмом прогонки. Выполнить оценку погрешности аппроксимации значений первой производной в узлах интерполяции.

1. Исходные данные

Вариант 3:



Вариант 18:



1. Решение

**Вариант 3**

Аппроксимация функции f(x) – замена этой функции некоторой другой функцией , близкой к исходной функции, но более простой с точки зрения проведения некоторых расчётов на основе этой функции, либо исследования функциональных свойств.

Одним из видов аппроксимации является интерполяция – способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. Обычно на практике применяется интерполяция многочленами, т.е. .

Пусть для функции  на отрезке  имеется таблица , где 

**Формула Ньютона**



Таблица разделенных разностей

1.000000 2.000000 9.493242 9.012163 5.703643 2.707303 1.028043

1.200000 3.898648 13.098107 12.434349 7.869485 3.735346

1.400000 6.518270 18.071847 17.156041 10.857762

1.600000 10.132639 24.934263 23.670698

1.800000 15.119492 34.402542

2.000000 22.000000

Интерполяционная формула Ньютона:



 - разделённая разность, определяемая следующими рекуррентными соотношениями:



Оценка погрешности для формулы Ньютона:



В нашем случае:



Оценка погрешности

x | f(x) | Pn(x) | Delta | Оценка |

1.1 | 2.873095 | 2.873332 | 0.000237 | 0.000570 |

1.3 | 5.103283 | 5.103200 | 0.000083 | 0.000190 |

1.5 | 8.180340 | 8.180402 | 0.000062 | 0.000136 |

1.7 | 12.425847 | 12.425756 | 0.000091 | 0.000190 |

1.9 | 18.283498 | 18.283783 | 0.000285 | 0.000570 |

**Интерполяция кубическим сплайном 1 дефекта**

Параметры  были вычислены с помощью алгоритма прогонки:





В нашем случае:



Оценка погрешности аппроксимации 1-й производной

x[i] | df/dx(x[i]) | m[i] | delta | оценка

1.0 | 8.047190 | 8.047190 | 0.000000 | 0.007199

1.2 | 11.102946 | 11.102176 | 0.000770 | 0.007199

1.4 | 15.319064 | 15.318150 | 0.000914 | 0.007199

1.6 | 21.136167 | 21.135083 | 0.001084 | 0.007199

1.8 | 29.162197 | 29.159846 | 0.002351 | 0.007199

2.0 | 40.235948 | 40.235948 | 0.000000 | 0.007199

Оценка погрешности аппроксимации функции

x | f(x) | S31(f;x) | Abs(f(x)-S31(f;x)) | Оценка

1.1 | 2.873095 | 2.872949 | 0.000145 | 0.001059

1.3 | 5.103283 | 5.103060 | 0.000223 | 0.001059

1.5 | 8.180340 | 8.180031 | 0.000309 | 0.001059

1.7 | 12.425847 | 12.425446 | 0.000400 | 0.001059

1.9 | 18.283498 | 18.282843 | 0.000655 | 0.001059



**Среднеквадратичное приближение (дискретный вариант)**

Система функции .

Тогда искомый полином запишется: .

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 6 | 9 | 14,2 |
|  | 9 | 14,2 | 23,4 |
|  | 14,2 | 23,4 | 39,9664 |

Найденные коэффициенты :





Вектор правых частей:

59.66904861

103.23126284

183.31657082

Оценка погрешности: 0.5236



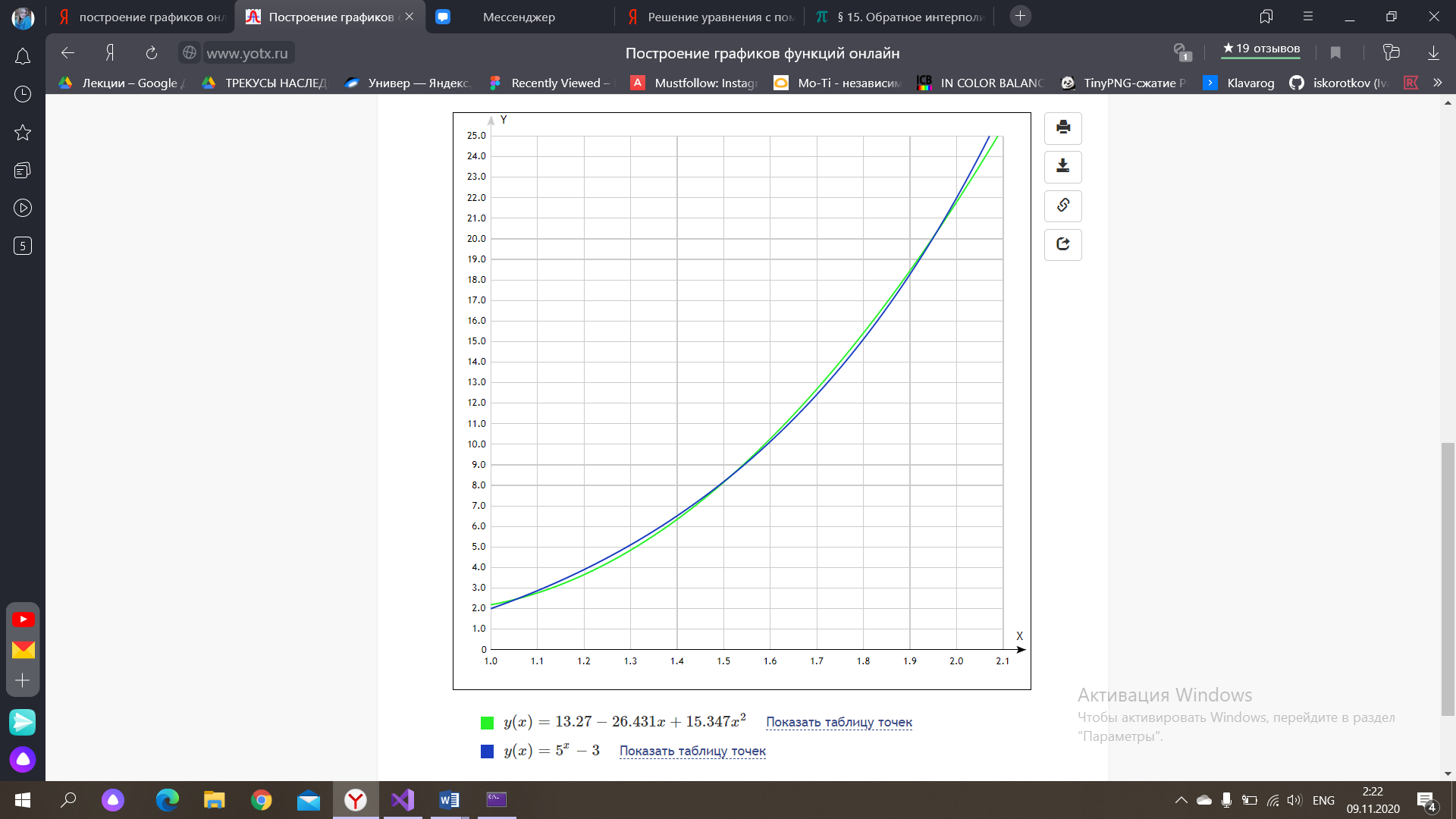


Рисунок 1 - График f(x) и g(x) на x=[1,2]

**Среднеквадратичное приближение (интегральный вариант):**

Система функции и вид искомой функции аналогичен дискретному варианту.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1.0 | 1.5 | 2.3333333 |
|  | 1.5 | 2.3333333 | 3.75 |
|  | 2.3333333 | 3.75 | 6.2 |





Вектор правых частей:

9.42669869

15.73893004

26.87651024

Оценка погрешности: 0.1529



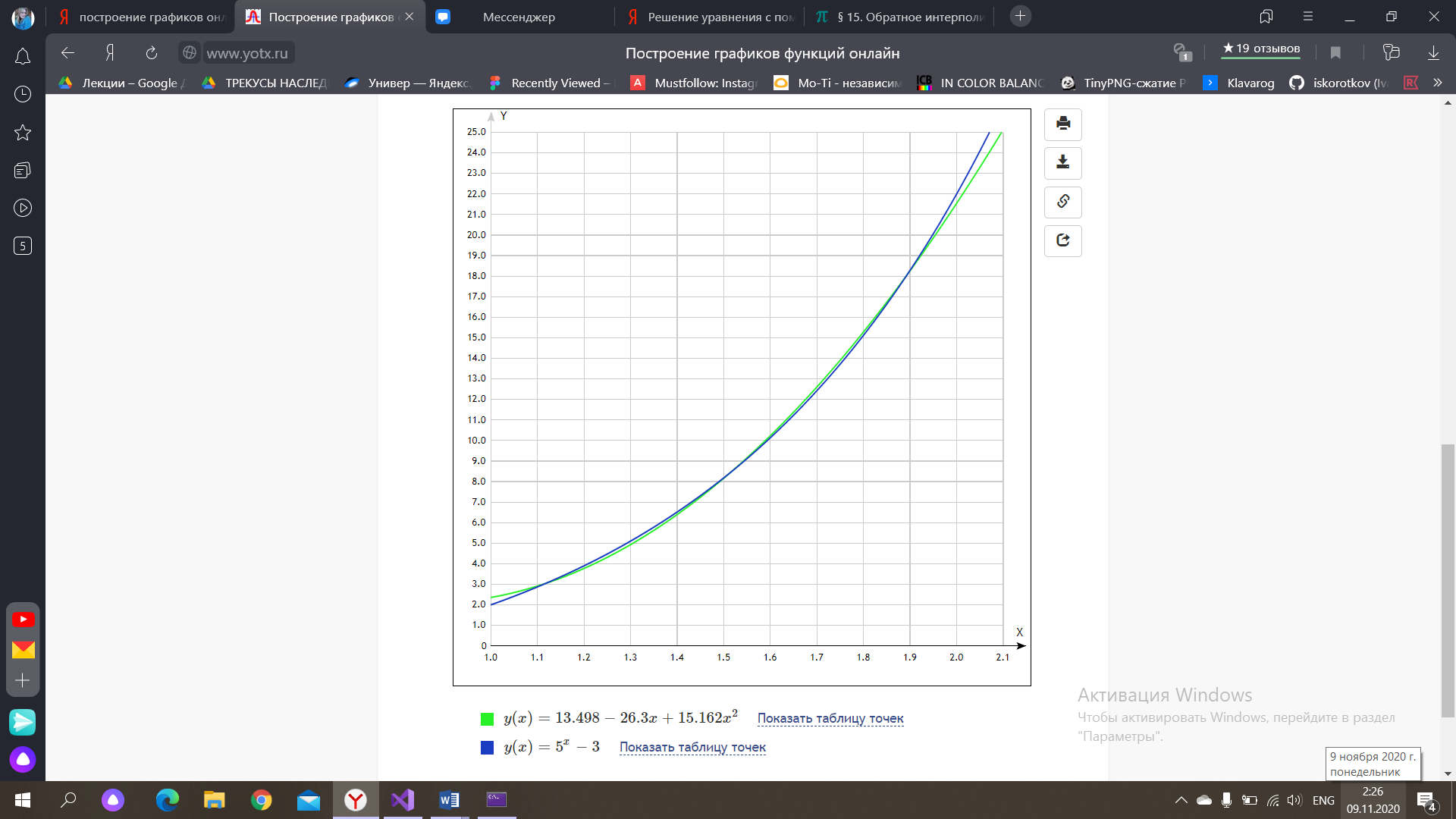


Рисунок 2 - График f(x) и g(x) на x=[1,2]

**Решение уравнения с помощью обратного интерполирования:**

Решаем уравнение 

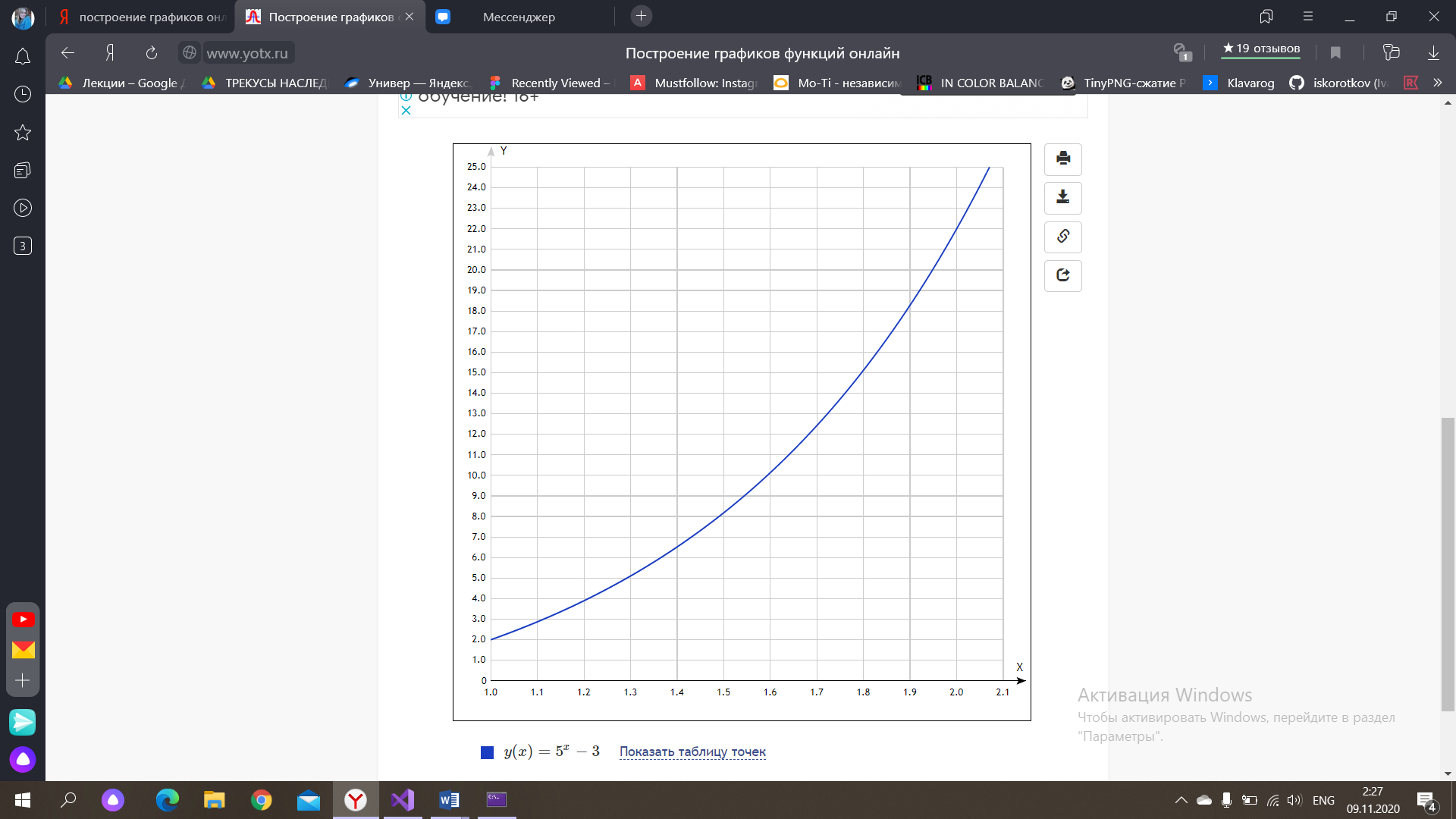


Рисунок 3 - График функции f(x) на [1,2]

Таким образом,

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 2.000000 | 1,0 |
| 3.898648 | 1,2 |
| 6.518270 | 1,4 |
| 10.132639 | 1,6 |
| 15.119492 | 1,8 |
| 22.000000 | 2,0 |

Таблица разделенных разностей

2.000000 1.000000 0.105338 -0.006416 0.000375 -0.000018 0.000001

3.898648 1.200000 0.076347 -0.003371 0.000143 -0.000005

6.518270 1.400000 0.055335 -0.001771 0.000054

10.132639 1.600000 0.040105 -0.000930

15.119492 1.800000 0.029068

22.000000 2.000000



и будет решением исходного уравнения.

Решим уравнение



Оценим погрешность по невязке:



**Вариант 18.**

**Формула Ньютона**



Таблица разделенных разностей

1.000000 6.000000 1.685964 2.264389 0.927384 0.284859 0.069999

1.200000 6.337193 2.591720 2.820819 1.155271 0.354857

1.400000 6.855537 3.720047 3.513981 1.439157

1.600000 7.599546 5.125640 4.377475

1.800000 8.624674 6.876630

2.000000 10.000000

Интерполяционная формула Ньютона:



 - разделённая разность, определяемая следующими рекуррентными соотношениями:



Оценка погрешности для формулы Ньютона:



В нашем случае:



Оценка погрешности

x | f(x) | Pn(x) | Delta | Оценка |

1.1 | 6.148370 | 6.148381 | 0.000011 | 0.000021 |

1.3 | 6.571168 | 6.571164 | 0.000004 | 0.000007 |

1.5 | 7.196152 | 7.196155 | 0.000003 | 0.000005 |

1.7 | 8.073008 | 8.073004 | 0.000004 | 0.000007 |

1.9 | 9.263626 | 9.263639 | 0.000013 | 0.000021 |

**Интерполяция кубическим сплайном 1 дефекта**

Параметры  были вычислены с помощью алгоритма прогонки:





В нашем случае:



Оценка погрешности аппроксимации 1-й производной

x[i] | df/dx(x[i]) | m[i] | delta | оценка

1.0 | 1.295837 | 1.295837 | 0.000000 | 0.000384

1.2 | 2.105726 | 2.105662 | 0.000064 | 0.000384

1.4 | 3.114630 | 3.114565 | 0.000065 | 0.000384

1.6 | 4.371453 | 4.371379 | 0.000074 | 0.000384

1.8 | 5.937116 | 5.936980 | 0.000136 | 0.000384

2.0 | 7.887511 | 7.887511 | 0.000000 | 0.000384

Оценка погрешности аппроксимации функции

x | f(x) | S31(f;x) | Abs(f(x)-S31(f;x)) | Оценка

1.1 | 6.148370 | 6.148351 | 0.000019 | 0.000074

1.3 | 6.571168 | 6.571142 | 0.000025 | 0.000074

1.5 | 7.196152 | 7.196121 | 0.000031 | 0.000074

1.7 | 8.073008 | 8.072970 | 0.000038 | 0.000074

1.9 | 9.263626 | 9.263574 | 0.000052 | 0.000074



**Среднеквадратичное приближение (дискретный вариант)**

Система функции .

Тогда искомый полином запишется: .

Таблица скалярных произведений

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 6 | 9 | 14,2 |
|  | 9 | 14,2 | 23,4 |
|  | 14,2 | 23,4 | 39,9664 |

Найденные коэффициенты :





Вектор правых частей:

45.41694973

70.88606991

115.96119168

Оценка погрешности: 0.0755





Рисунок 4 - График f(x) и g(x) на x=[0;3]

**Среднеквадратичное приближение (интегральный вариант):**

Система функции и вид искомой функции аналогичен дискретному варианту.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1.0 | 1.5 | 2.3333333 |
|  | 1.5 | 2.3333333 | 3.75 |
|  | 2.3333333 | 3.75 | 6.2 |





Вектор правых частей:

7.46143536

11.51570903

18.39848326

Оценка погрешности:





Рисунок 5 - График f(x) и g(x) на x=[0;3]

**Решение уравнения с помощью обратного интерполирования:**

Решаем уравнение 



Рисунок 6 - График функции f(x) на [1;2]

Таким образом,

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 6,00000000 | 1,0 |
| 6,33719282 | 1,2 |
| 6,85553672 | 1,4 |
| 7,59954613 | 1,6 |
| 8,62467406 | 1,8 |
| 10,00000000 | 2,0 |

Таблица разделенных разностей

6.000000 1.000000 0.593132 -0.242290 0.093515 -0.027128 0.005714

6.337193 1.200000 0.385844 -0.092708 0.022313 -0.004271

6.855537 1.400000 0.268814 -0.041668 0.006670

7.599546 1.600000 0.195098 -0.020695

8.624674 1.800000 0.145420

10.000000 2.000000



и будет решением исходного уравнения.

Решим уравнение



Оценим погрешность по невязке:



1. Тестирование

Протестируем программу на примере двух матриц из двух вариантов.

**Вариант 3:**

Интерполяционная формула Ньютона

Таблица разделенных разностей:

1.000000 2.000000 9.493242 9.012163 5.703643 2.707303 1.028043

1.200000 3.898648 13.098107 12.434349 7.869485 3.735346

1.400000 6.518270 18.071847 17.156041 10.857762

1.600000 10.132639 24.934263 23.670698

1.800000 15.119492 34.402542

2.000000 22.000000

M6 = 434.495591

x | f(x) | Pn(x) | Delta | Оценка |

1.1 | 2.873095 | 2.873332 | 0.000237 | 0.000570 |

1.3 | 5.103283 | 5.103200 | 0.000083 | 0.000190 |

1.5 | 8.180340 | 8.180402 | 0.000062 | 0.000136 |

1.7 | 12.425847 | 12.425756 | 0.000091 | 0.000190 |

1.9 | 18.283498 | 18.283783 | 0.000285 | 0.000570 |

Интерполяция кубическим сплайном

M5 = 269.967289

M4 = 167.740108

x[i] | df/dx(x[i]) m[i] delta оценка

1.0 | 8.047190 | 8.047190 | 0.000000 | 0.007199

1.2 | 11.102946 | 11.102176 | 0.000770 | 0.007199

1.4 | 15.319064 | 15.318150 | 0.000914 | 0.007199

1.6 | 21.136167 | 21.135083 | 0.001084 | 0.007199

1.8 | 29.162197 | 29.159846 | 0.002351 | 0.007199

2.0 | 40.235948 | 40.235948 | 0.000000 | 0.007199

x | f(x) S31(f;x) Abs(f(x)-S31(f;x)) Оценка

1.1 | 2.873095 | 2.872949 | 0.000145 | 0.001059

1.3 | 5.103283 | 5.103060 | 0.000223 | 0.001059

1.5 | 8.180340 | 8.180031 | 0.000309 | 0.001059

1.7 | 12.425847 | 12.425446 | 0.000400 | 0.001059

1.9 | 18.283498 | 18.282843 | 0.000655 | 0.001059

Среднеквадратичное приближение

Дискретный вариант

Матрица:

6.0000 9.0000 14.2000

9.0000 14.2000 23.4000

14.2000 23.4000 39.9664

Вектор правых частей:

59.66904861

103.23126284

183.31657082

P2(x) = 13.2695 + (-26.4313\*x) + (15.3474\*x^2)

Оценка погрешности: sqrt(876.9567 - 876.6825) = 0.5236

Непрерывный вариант

Матрица:

1.0000 1.5000 2.3333

1.5000 2.3333 3.7500

2.3333 3.7500 6.2000

Вектор правых частей:

9.42669869

15.73893004

26.87651024

P2(x) = 13.4981 + (-26.2998\*x) + (15.1621\*x^2)

Оценка погрешности: sqrt(120.8403 - 120.8169) = 0.1529

Обратная интерполяция

Таблица разделенных разностей:

2.000000 1.000000 0.105338 -0.006416 0.000375 -0.000018 0.000001

3.898648 1.200000 0.076347 -0.003371 0.000143 -0.000005

6.518270 1.400000 0.055335 -0.001771 0.000054

10.132639 1.600000 0.040105 -0.000930

15.119492 1.800000 0.029068

22.000000 2.000000

C= 8.180340

Корень: 1.499603

Невязка: 0.00713919

**Вариант 18:**

Интерполяционная формула Ньютона

Таблица разделенных разностей:

1.000000 6.000000 1.685964 2.264389 0.927384 0.284859 0.069999

1.200000 6.337193 2.591720 2.820819 1.155271 0.354857

1.400000 6.855537 3.720047 3.513981 1.439157

1.600000 7.599546 5.125640 4.377475

1.800000 8.624674 6.876630

2.000000 10.000000

M6 = 15.823743

x | f(x) | Pn(x) | Delta | Оценка |

1.1 | 6.148370 | 6.148381 | 0.000011 | 0.000021 |

1.3 | 6.571168 | 6.571164 | 0.000004 | 0.000007 |

1.5 | 7.196152 | 7.196155 | 0.000003 | 0.000005 |

1.7 | 8.073008 | 8.073004 | 0.000004 | 0.000007 |

1.9 | 9.263626 | 9.263639 | 0.000013 | 0.000021 |

Интерполяция кубическим сплайном

M5 = 14.403392

M4 = 13.110532

x[i] | df/dx(x[i]) m[i] delta оценка

1.0 | 1.295837 | 1.295837 | 0.000000 | 0.000384

1.2 | 2.105726 | 2.105662 | 0.000064 | 0.000384

1.4 | 3.114630 | 3.114565 | 0.000065 | 0.000384

1.6 | 4.371453 | 4.371379 | 0.000074 | 0.000384

1.8 | 5.937116 | 5.936980 | 0.000136 | 0.000384

2.0 | 7.887511 | 7.887511 | 0.000000 | 0.000384

x | f(x) S31(f;x) Abs(f(x)-S31(f;x)) Оценка

1.1 | 6.148370 | 6.148351 | 0.000019 | 0.000074

1.3 | 6.571168 | 6.571142 | 0.000025 | 0.000074

1.5 | 7.196152 | 7.196121 | 0.000031 | 0.000074

1.7 | 8.073008 | 8.072970 | 0.000038 | 0.000074

1.9 | 9.263626 | 9.263574 | 0.000052 | 0.000074

Среднеквадратичное приближение

Дискретный вариант

Матрица:

6.0000 9.0000 14.2000

9.0000 14.2000 23.4000

14.2000 23.4000 39.9664

Вектор правых частей:

45.41694973

70.88606991

115.96119168

P2(x) = 8.5279 + (-5.7229\*x) + (3.2222\*x^2)

Оценка погрешности: sqrt(355.2965 - 355.2908) = 0.0755

Непрерывный вариант

Матрица:

1.0000 1.5000 2.3333

1.5000 2.3333 3.7500

2.3333 3.7500 6.2000

Вектор правых частей:

7.46143536

11.51570903

18.39848326

P2(x) = 8.5792 + (-5.7290\*x) + (3.2039\*x^2)

Оценка погрешности: sqrt(56.9868 - 56.9863) = 0.0221

Обратная интерполяция

Таблица разделенных разностей:

Таблица разделенных разностей:

6.000000 1.000000 0.593132 -0.242290 0.093515 -0.027128 0.005714

6.337193 1.200000 0.385844 -0.092708 0.022313 -0.004271

6.855537 1.400000 0.268814 -0.041668 0.006670

7.599546 1.600000 0.195098 -0.020695

8.624674 1.800000 0.145420

10.000000 2.000000

C= 7.196152

Корень: 1.498246

Невязка: 0.00649619

1. Краткие выводы

В ходе выполнения данной лабораторной работы мы научились интерполировать и находить приближения функций различными методами.

1. Текст программы.

#include <windows.h>

#include "iostream"

#include "cmath"

#include <consoleapi2.h>

#include <iomanip>

#include <vector>

using namespace std;

const int var = 18;

const int n = 6;

double b = 2.0;

double a = 1.0;

double h = 1 / 5.;

typedef vector<double> myVector;

typedef vector<myVector> myMatrix;

double\* yArr = new double[n];

//исходная функция

double F(double x)

{

if (var == 11)

return pow(3, x) + 2 \* x - 2;

if (var == 3)

return pow(5, x) - 3;

else

return pow(3, x) - 2 \* x + 5;

}

//производная функции

double DF(double x)

{

if (var == 11)

return log(3) \* pow(3, x) + 2;

if (var == 3)

return log(5) \* pow(5, x);

else

return log(3) \* pow(3, x) - 2;

}

void get\_X\_Y\_Arr(double\* xArr, double\* xArr1, double\* yArr)

{

double a = 1.0;

for (int i = 0; i < n; i++)

{

xArr[i] = 1.0 + i \* h;

}

for (int i = 0; i < n - 1; i++)

{

xArr1[i] = 1.0 + (i + 0.5) \* h;

}

for (int i = 0; i < n; i++)

{

yArr[i] = F(xArr[i]);

}

}

// построение таблицы разделенных разностей

void FindDividedDifferenceMatrix(double\*\* DivDiff, double\* xArr, double\* yArr)

{

for (int i = 0; i < n; i++)

{

DivDiff[i][0] = xArr[i];

DivDiff[i][1] = yArr[i];

}

for (int j = 2; j <= n; j++)

{

for (int i = 0; i < n - j + 1; i++)

DivDiff[i][j] = (DivDiff[i + 1][j - 1] - DivDiff[i][j - 1]) / (xArr[i + j - 1] - xArr[i]);

}

}

double fi0(double tau)

{

return (1 + 2 \* tau) \* pow(1 - tau, 2);

}

double fi1(double tau)

{

return tau \* pow(1 - tau, 2);

}

//погрешность интерполяции сплайнами

double FindErrorForSpline(double M4, double M5)

{

return (M4 / 384 + M5 \* h / 240) \* pow(h, 4);

}

// Метод прогонки

myVector SweepMethod(myMatrix m, myVector r, int const n)

{

double y;

myVector a(n + 1), B(n + 1), resV(n + 1);

y = m[0][0];

a[0] = -m[0][1] / y;

B[0] = r[0] / y;

for (int i = 1; i < n; i++) {

y = m[i][i] + m[i][i - 1] \* a[i - 1];

a[i] = -m[i][i + 1] / y;

B[i] = (r[i] - m[i][i - 1] \* B[i - 1]) / y;

}

resV[n] = (r[n] - m[n][n - 1] \* B[n - 1]) / (m[n][n] + m[n][n - 1] \* a[n - 1]);

for (int i = n - 1; i >= 0; i--)

{

resV[i] = a[i] \* resV[i + 1] + B[i];

}

return resV;

}

//Интерполяция кубическим сплайном

void SplineInterpolation(double\* xArr1, double\* xArr2)

{

cout << "\nИнтерполяция кубическим сплайном\n";

double\* m = new double[n];

double df0 = DF(1), dfn = DF(2);

m[0] = df0;

m[n - 1] = dfn;

// находим параметры m методом прогонки

double\* alpha = new double[n];

double\* beta = new double[n];

alpha[1] = 0;

beta[1] = df0;

for (int j = 1; j < n - 1; j++)

{

alpha[j + 1] = -1 / (4 + alpha[j]);

beta[j + 1] = (3 \* (F(xArr1[j + 1]) - F(xArr1[j - 1])) / h - beta[j]) / (4 + alpha[j]);

}

for (int j = n - 2; j >= 0; j--)

m[j] = alpha[j + 1] \* m[j + 1] + beta[j + 1];

/\*for (int j = 1; j < n - 1; j++)

m[j] = (F(xArr1[j + 1]) - F(xArr1[j - 1])) / (2 \* h);\*/

//наибольшее значение 4-й и 5-й производной достигается в точке х = 2

double M4, M5;

M4 = (var == 3) ? pow(log(5), 4) \* 25 : pow(log(3), 4) \* 9;

M5 = (var == 3) ? pow(log(5), 5) \* 25 : pow(log(3), 5) \* 9;

cout << "M5 = " << M5 << "\nM4 = " << M4 << endl << endl;

// оценка погрешности аппроксимации 1-й производной

cout << " x[i] | " << " df/dx(x[i]) " << " m[i] " << " delta " << " оценка " << endl;

for (int i = 0; i < n; i++)

{

double DF\_xi = DF(xArr1[i]);

cout << setprecision(1) << xArr1[i] << " | " << setprecision(6) << DF\_xi << " | " << m[i] << " | " <<

abs(DF\_xi - m[i]) << " | " << M5 / 60 \* pow(h, 4) << endl;

}

// оценка погрешности аппроксимации функции

cout << endl << " x | " << " f(x) " << " S31(f;x) " << " Abs(f(x)-S31(f;x)) " << " Оценка " << endl;

for (int i = 0; i < n - 1; i++)

{

double tau = 0.5; // (xArr2[i] - xArr1[i]) / h;

double S = fi0(tau) \* F(xArr1[i]) + fi0(1 - tau) \* F(xArr1[i + 1]) +

h \* (fi1(tau) \* m[i] - fi1(1 - tau) \* m[i + 1]);

double F\_ = F(xArr2[i]);

cout << setprecision(1) << xArr2[i] << " | " << setprecision(6) << F\_ << " | " <<

S << " | " << abs(F\_ - S) << " | " << FindErrorForSpline(M4, M5) << endl;

}

//delete[] b1;

}

//печать Таблицы разделенных разностей

void PrintDividedDifferenceMatrix(double\*\* DivDiff)

{

cout << "Таблица разделенных разностей:\n";

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= n - i; j++)

cout << fixed << setprecision(6) << setw(12) << DivDiff[i][j];

cout << endl;

}

}

// погрешность интерполяции по формуле Ньютона

double FindErrorForNewton(double x, double\* xArr, double M6)

{

double w = 1;

for (int i = 0; i < n; i++)

w \*= x - xArr[i];

int fact = 1;

for (int i = 2; i < n + 1; i++)

fact \*= i;

return M6 \* abs(w) / fact;

}

//Newton

void Newton(double\* xArr, double\* xArr1, double\* yArr)

{

cout << "Интерполяционная формула Ньютона \n";

double\*\* DivDiff = new double\* [n];

for (int i = 0; i < n; i++)

{

DivDiff[i] = new double[n + 1];

}

get\_X\_Y\_Arr(xArr, xArr1, yArr);

FindDividedDifferenceMatrix(DivDiff, xArr, yArr);

PrintDividedDifferenceMatrix(DivDiff);

double M6 = (var == 3) ? 25 \* pow(log(5), 6) : 9 \* pow(log(3), 6);

//double M6 = pow(log(3), 6) \* 9;

cout << "\nM6 = " << M6 << endl;

cout << "\nx | f(x) | Pn(x) | Delta | Оценка |\n";

for (int k = 0; k < n - 1; k++)

{

double Pn\_x = DivDiff[0][1];

double w = 1;

for (int i = 1; i < n; i++)

{

w \*= xArr1[k] - xArr[i - 1];

Pn\_x += DivDiff[0][i + 1] \* w;

}

double F\_ = F(xArr1[k]);

cout << setprecision(1) << xArr1[k] << " | " << setprecision(6) << F\_ << " | " <<

Pn\_x << " | " << abs(Pn\_x - F\_) << " | " << FindErrorForNewton(xArr1[k], xArr, M6) << " | \n";

}

for (int i = 0; i < n; i++)

delete[] DivDiff[i];

delete[] DivDiff;

}

// печать матрицы

void PrintMatrix(double\*\* A)

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

cout << fixed << setprecision(4) << setw(10) << A[i][j];

cout << endl;

}

cout << "\n";

}

// печать вектора

void PrintVector(double\* b)

{

for (int i = 0; i < 3; i++)

cout << setprecision(8) << b[i] << endl;

cout << "\n";

}

// определитель

double Determinant(double\*\* A)

{

return A[0][0] \* A[1][1] \* A[2][2] +

A[0][1] \* A[1][2] \* A[2][0] +

A[0][2] \* A[1][0] \* A[2][1] -

A[0][2] \* A[1][1] \* A[2][0] -

A[0][0] \* A[2][1] \* A[1][2] -

A[2][2] \* A[1][0] \* A[0][1];

}

// замена столбца матрицы на вектор

void ReplaceColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

A[i][j] = b[i];

}

// копирование столбца матрицы

void CopyColumn(double\*\* A, int j, double\* b)

{

for (int i = 0; i < 3; i++)

b[i] = A[i][j];

}

// решение СЛАУ методом Крамера

void CramerMethod(double\*\* A, double\* b, double\* x)

{

double det = Determinant(A);

double\* aj = new double[3];

CopyColumn(A, 0, aj);

ReplaceColumn(A, 0, b);

double det1 = Determinant(A);

ReplaceColumn(A, 0, aj);

CopyColumn(A, 1, aj);

ReplaceColumn(A, 1, b);

double det2 = Determinant(A);

ReplaceColumn(A, 1, aj);

CopyColumn(A, 2, aj);

ReplaceColumn(A, 2, b);

double det3 = Determinant(A);

x[0] = det1 / det;

x[1] = det2 / det;

x[2] = det3 / det;

}

// функция, получаемая методом среднеквадратичного приближения

double FuncBySquare(double x, double\* c)

{

return c[0] + c[1] \* x + c[2] \* pow(x, 2);

}

// среднеквадратичное приближение (дискретный вариант)

void MeanSquareApproximationForTable(double\* xArr1, double\* xArr2)

{

cout << "Дискретный вариант\n\n";

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

A[i][j] = 0;

for (int i = 0; i < n; i++)

{

A[0][0] ++;

A[0][1] += xArr1[i];

A[0][2] += pow(xArr1[i], 2);

A[1][2] += pow(xArr1[i], 3);

A[2][2] += pow(xArr1[i], 4);

}

A[1][0] = A[0][1];

A[1][1] = A[0][2];

A[2][0] = A[0][2];

A[2][1] = A[1][2];

cout << "Матрица:\n";

PrintMatrix(A);

double\* b = new double[3];

for (int i = 0; i < 3; i++)

b[i] = 0;

for (int i = 0; i < n; i++)

{

b[0] += F(xArr1[i]);

b[1] += F(xArr1[i]) \* xArr1[i];

b[2] += F(xArr1[i]) \* pow(xArr1[i], 2);

}

cout << "\nВектор правых частей:\n";

PrintVector(b);

double\* c = new double[3];

CramerMethod(A, b, c);

cout << setprecision(4) << "P2(x) = " << c[0] << " + (" << c[1] << "\*x) + (" << c[2] << "\*x^2)\n";

double scalarFF = 0; // (f,f)

double scalarFG = 0; // (f,g[i])

for (int i = 0; i < n; i++)

{

scalarFF += pow(F(xArr1[i]), 2);

scalarFG += pow(FuncBySquare(xArr1[i], c), 2);

}

double error = sqrt(scalarFF - scalarFG);

cout << "Оценка погрешности: " << "sqrt(" << scalarFF << " - " << scalarFG << ") = " << error << endl;

for (int i = 0; i < 3; i++)

delete[] A[i];

delete[] A;

delete[] b;

delete[] c;

}

double g(double x, int n)

{

switch (n)

{

case 0: return 1;

case 1: return x;

case 2: return x \* x;

}

}

// вычисляем интеграл по формуле трапеций

double IntegrG(double a, double b, int n)

{

double eps = 0.000001;

double Integral = eps \* (F(a) \* g(a, n) + F(b) \* g(b, n)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* F(a + eps \* i) \* g(a + eps \* i, n);

return Integral;

}

// вычисляем интеграл по формуле трапеций

double IntegrF(double a, double b)

{

double eps = 0.000001;

double Integral = eps \* (F(a) \* F(a) + F(b) \* F(b)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* F(a + eps \* i) \* F(a + eps \* i);

return Integral;

}

double G(myVector B, double x)

{

return B[0] + B[1] \* x + B[2] \* x \* x;

}

double IntegrGG(double a, double b, myVector B)

{

// вычисляем интеграл по формуле трапеций

double eps = 0.000001;

double Integral = eps \* (G(B, a) \* G(B, a) + G(B, b) \* G(B, b)) / 2.0;

for (int i = 1; i <= (b - a) / eps - 1; i++)

Integral = Integral + eps \* G(B, a + eps \* i) \* G(B, a + eps \* i);

return Integral;

}

// среднеквадратичное приближение (непрерывный вариант)

void MeanSquareApproximationForInterval(double\* xArr2)

{

cout << "\nНепрерывный вариант\n\n";

double\*\* A = new double\* [3];

for (int i = 0; i < 3; i++)

A[i] = new double[3];

A[0][0] = 1;

A[0][1] = 3 / 2.;

A[0][2] = 7 / 3.;

A[1][0] = 3 / 2.;

A[1][1] = 7 / 3.;

A[1][2] = 15 / 4.;

A[2][0] = 7 / 3.;

A[2][1] = 15 / 4.;

A[2][2] = 31 / 5.;

cout << "Матрица:\n";

PrintMatrix(A);

double\* r = new double[3];

r[0] = IntegrG(a, b, 0);

r[1] = IntegrG(a, b, 1);

r[2] = IntegrG(a, b, 2);

cout << "\nВектор правых частей:\n";

PrintVector(r);

double\* c = new double[3];

CramerMethod(A, r, c);

cout << setprecision(4) << "P2(x) = " << c[0] << " + (" << c[1] << "\*x) + (" << c[2] << "\*x^2)\n";

double F = 0;

F = IntegrF(a, b) \* IntegrF(a, b);

cout << "\nОценка погрешности: " << "sqrt(" << IntegrF(a, b) << " - " << IntegrGG(a, b, { c[0], c[1], c[2] }) << ") = " << sqrt(IntegrF(a, b) - IntegrGG(a, b, { c[0], c[1], c[2] })) << endl << endl;

}

void ReverseInterpolation(double\* xArr, double\* yArr)

{

// таблица разделенных разностей

double\*\* DivDiff = new double\* [n];

for (int i = 0; i < n; i++)

DivDiff[i] = new double[n + 1];

cout << "\nОбратная интерполяция\n\n";

FindDividedDifferenceMatrix(DivDiff, yArr, xArr);

cout << "Таблица разделенных разностей:\n";

PrintDividedDifferenceMatrix(DivDiff);

double x = DivDiff[0][1];

double w = 1;

double c = F(1.5); // значение функции в середине отрезка

for (int i = 1; i < n; i++)

{

w \*= c - yArr[i - 1];

x += DivDiff[0][i + 1] \* w;

}

cout << "C= " << c << endl;

cout << "\nКорень: " << x << endl;

cout << "Невязка: " << setprecision(8) << abs(F(x) - c) << endl;

for (int i = 0; i < n; i++)

delete[] DivDiff[i];

delete[] DivDiff;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

double\* xArr = new double[n];

double\* xArr1 = new double[n];

get\_X\_Y\_Arr(xArr, xArr1, yArr);

Newton(xArr, xArr1, yArr);

SplineInterpolation(xArr, xArr1);

cout << "\n Среднеквадратичное приближение\n\n";

MeanSquareApproximationForTable(xArr, xArr1);

MeanSquareApproximationForInterval(xArr1);

ReverseInterpolation(xArr, yArr);

cout << endl;

delete[] xArr;

delete[] xArr1;

delete[] yArr;

system("pause");

}