

Дисциплина: Численные методы

Лабораторное задание №1

Отчет

Тема: «Интерполирование и приближение функций»

Выполнил:

студент 3 курса 8 группы

Крутько А.С.

Проверила:

преподаватель

Махинова О.А.

Первая задача. Интерполирование функций алгебраическими многочленами.....	3
Постановка задачи	3
Теоретическая часть.....	4
Вычислительный эксперимент.....	5
Тестирование	8
Вывод.....	14
Вторая задача. Аппроксимация табличной функции (метод наименьших квадратов)	15
Постановка задачи	15
Теоретические сведения.....	16
Вычислительный эксперимент.....	17
Вывод.....	20
Третья задача. Интерполирование функций кубическими сплайнами.....	21
Постановка задачи	21
Теоретические сведения.....	22
Вычислительный эксперимент.....	26
Вывод.....	28

Первая задача. Интерполирование функций алгебраическими многочленами

Постановка задачи

Построить полином в форме Ньютона, выполнив построение по формулам для коэффициентов b_i .

Входные параметры основной функции:

- $f(x)$ — интерполируемая функция;
- $[a; b]$ — отрезок разбиения
- n — количество отрезков разбиения

Теоретическая часть

Построение полинома в форме Ньютона заключается в подсчёте такого численного значения как разделенные разности. В моём же случае данные значения нужно было находить по следующей формуле:

$$b_k = \sum_{i=0}^k (-1)^{k-i} \frac{f_i}{i! (k-i)! h^k}$$

В программе подсчёт данных коэффициентов и получение полинома было реализовано следующим образом:

```
private static double CalculateDividedDifferences(
    IReadOnlyList<double> x,
    IReadOnlyList<double> y,
    int k
)
{
    var result = 0.0;

    var h:double = x[1] - x[0];

    for (var i = 0; i ≤ k; i++)
    {
        result += Math.Pow(-1, k - i) * y[i]
                /
                (MathFunctions.Fact(i) * MathFunctions.Fact(n:k - i) * Math.Pow(h, k));
    }
    return result;
}
```

Рисунок 1 Метод для подсчёта разделенных разностей

```
private static Func<double, double> CreateNewtonPolynomial(double[] x, double[] y)
{
    var divDiff = new double[x.Length - 1];
    for (var index = 1; index < x.Length; index++)
    {
        divDiff[index - 1] = CalculateDividedDifferences(x, y, index);
    }

    double NewtonPolynomial(double xVal)
    {
        var result:double = y[0];
        for (var index = 1; index < x.Length; index++)
        {
            double mul = 1;
            for (var innerIndex = 0; innerIndex < index; innerIndex++)
            {
                mul *= (xVal - x[innerIndex]);
            }

            result += divDiff[index - 1] * mul;
        }

        return result;
    }

    return NewtonPolynomial;
}
```

Рисунок 2 Метод для задания интерполяционного полинома

Вычислительный эксперимент

Исследовать зависимость погрешности интерполяции от степени интерполяционного полинома. Для нескольких функций сформировать таблицу погрешностей:

- В первом столбце – степень интерполяционного полинома;
- Во втором столбце – погрешность интерполяции при равномерном разбиении отрезка $[a; b]$;
- В третьем столбце – погрешность интерполяции при разбиении Чебышева отрезка $[a; b]$;

1. Задана функция $f(x) = x$ на отрезке $[-3, 3]$

Количество точек разбиения	Погрешность интерполяции при равномерном разбиении отрезка $[a, b]$	Погрешность интерполяции при разбиении Чебышева отрезка $[a, b]$
2	0	2.60398256776706E-16
4	0	5.58570184480636E-16
8	3.90798504668055E-14	5.90638649100583E-14
16	8.42081959717689E-12	8.56519744019124E-10
32	0.00027166282091029	0.0770776083288793

2. Задана функция $f(x) = \sin(x)$ на отрезке $[-3, 3]$

Количество точек разбиения	Погрешность интерполяции при равномерном разбиении отрезка $[a, b]$	Погрешность интерполяции при разбиении Чебышева отрезка $[a, b]$
2	0	3.83536518516747E-17
4	9.99200722162641E-16	9.99200722162641E-16
8	2.98649993624167E-14	5.14033260401447E-14
16	6.39077679664979E-11	9.92300686064596E-11
32	0.00296367361578159	0.00860714938993234

Для нескольких заданных функций построить графики:

1. Функция $y = x^3$ на отрезке $[-3; 3]$

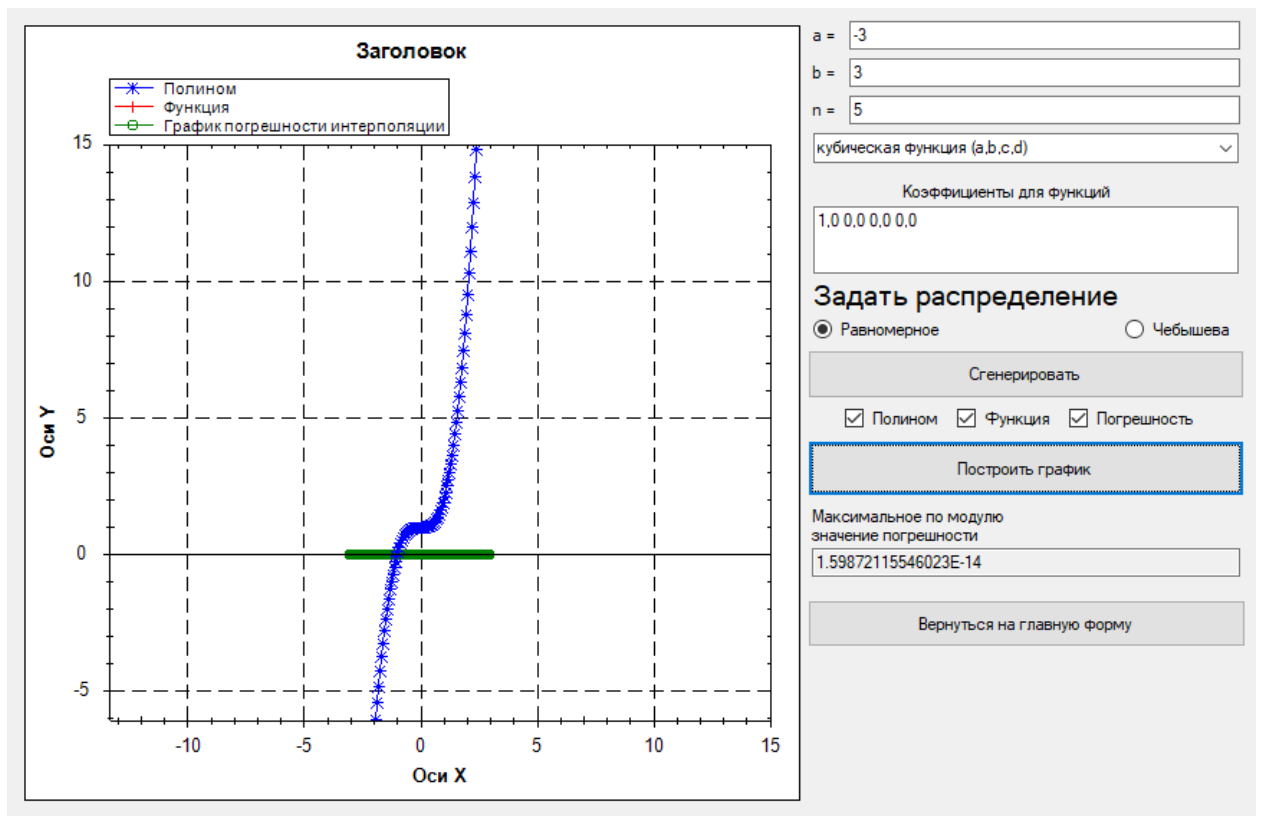


Рисунок 3 Функция $f(x) = x^3$ на отрезке $[-3; 3]$

Количество значений в узлах интерполяции соответствует количеству вхождений алгоритма при поиске значений полинома P , что соответствует его степени.

Это можно проследить по окну вывода после работы метода, дающего значения полинома в определенной точке

```

Вхождение для элемента с индексом суммы 1
Вхождение для элемента с индексом произведения 0
Вхождение для элемента с индексом суммы 2
Вхождение для элемента с индексом произведения 0
Вхождение для элемента с индексом произведения 1
Вхождение для элемента с индексом суммы 3
Вхождение для элемента с индексом произведения 0
Вхождение для элемента с индексом произведения 1
Вхождение для элемента с индексом произведения 2
Вхождение для элемента с индексом суммы 4
Вхождение для элемента с индексом произведения 0
Вхождение для элемента с индексом произведения 1
Вхождение для элемента с индексом произведения 2
Вхождение для элемента с индексом произведения 3
Вхождение для элемента с индексом суммы 5
Вхождение для элемента с индексом произведения 0
Вхождение для элемента с индексом произведения 1
Вхождение для элемента с индексом произведения 2
Вхождение для элемента с индексом произведения 3
Вхождение для элемента с индексом произведения 4

```

Соответственно код метода:

```

double NewtonPolynomial(double xVal)
{
    var result:double = divDiff[0];
    for (var index = 1; index < x.Length; index++)
    {
        double mul = 1;

        Console.WriteLine(
            "Вхождение для элемента с индексом суммы {0}",
            index
        );

        for (var innerIndex = 0; innerIndex < index; innerIndex++)
        {
            mul *= xVal - x[innerIndex];
            Console.WriteLine(
                "Вхождение для элемента с индексом произведения {0}",
                innerIndex
            );
        }

        result += divDiff[index] * mul;
    }

    return result;
}

```

Как можно заметить, получается действительно полином степени 5.

Тестирование

- Зафиксировать n и подобрать несколько тестовых примеров, демонстрирующих что строится действительно интерполяционный полином степени n .

Зафиксируем $n = 8$ на отрезке $[0; 3]$

1. Функция $f(x) = 1 + \sin(x)$

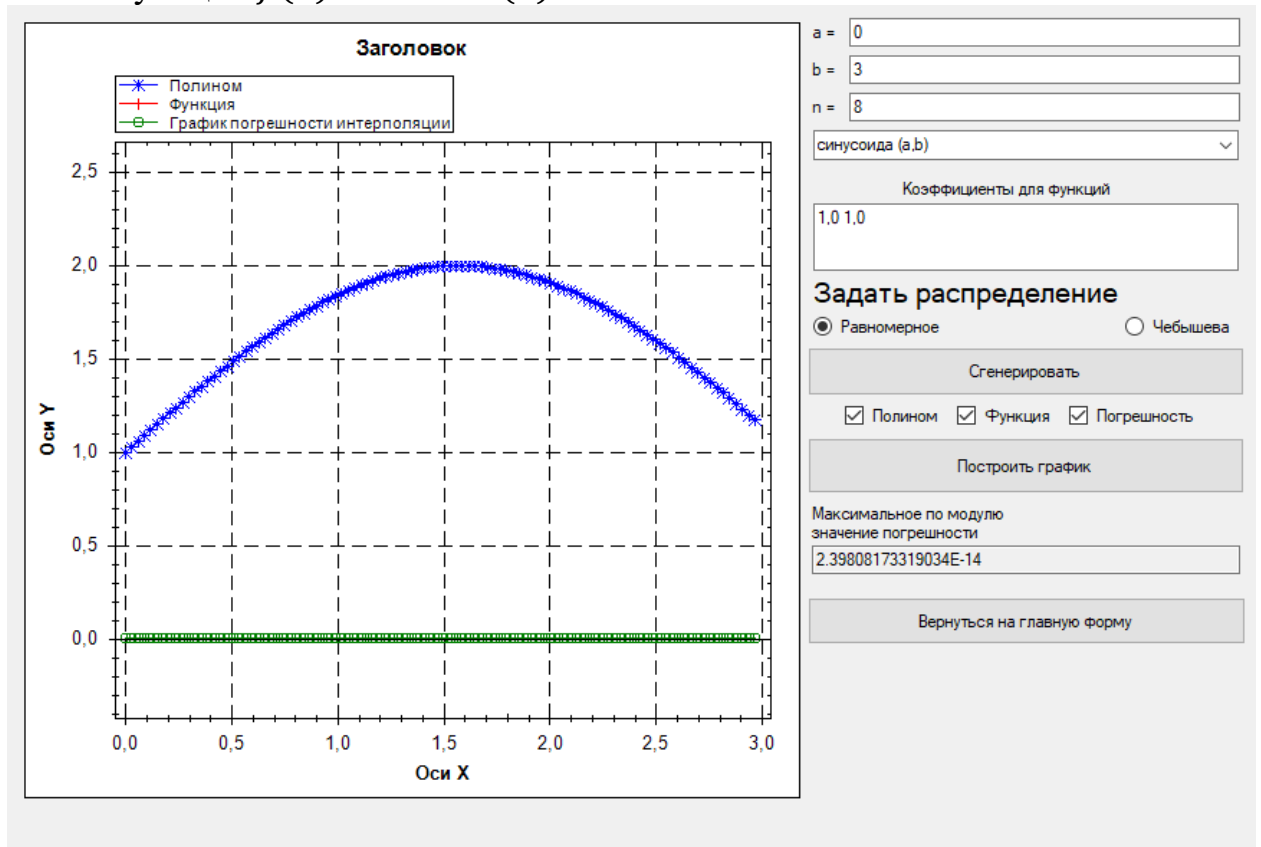


Рисунок 4 Функция $f(x) = 1 + \sin(x)$

Значения функции Y:

Индекс	Значение
[0]	1
[1]	1.3662725290860476
[2]	1.681638760023334
[3]	1.9022675940990952
[4]	1.9974949866040546
[5]	1.9540857816096939
[6]	1.7780731968879211
[7]	1.4939202986100892

Значения полинома P:

Индекс	Значение
[0]	1
[1]	1.3662725290860478
[2]	1.6816387600233345
[3]	1.9022675940990959
[4]	1.9974949866040559
[5]	1.9540857816096968
[6]	1.7780731968879291
[7]	1.4939202986101132

2. Функция $f(x) = x^2 + 2x + 1$

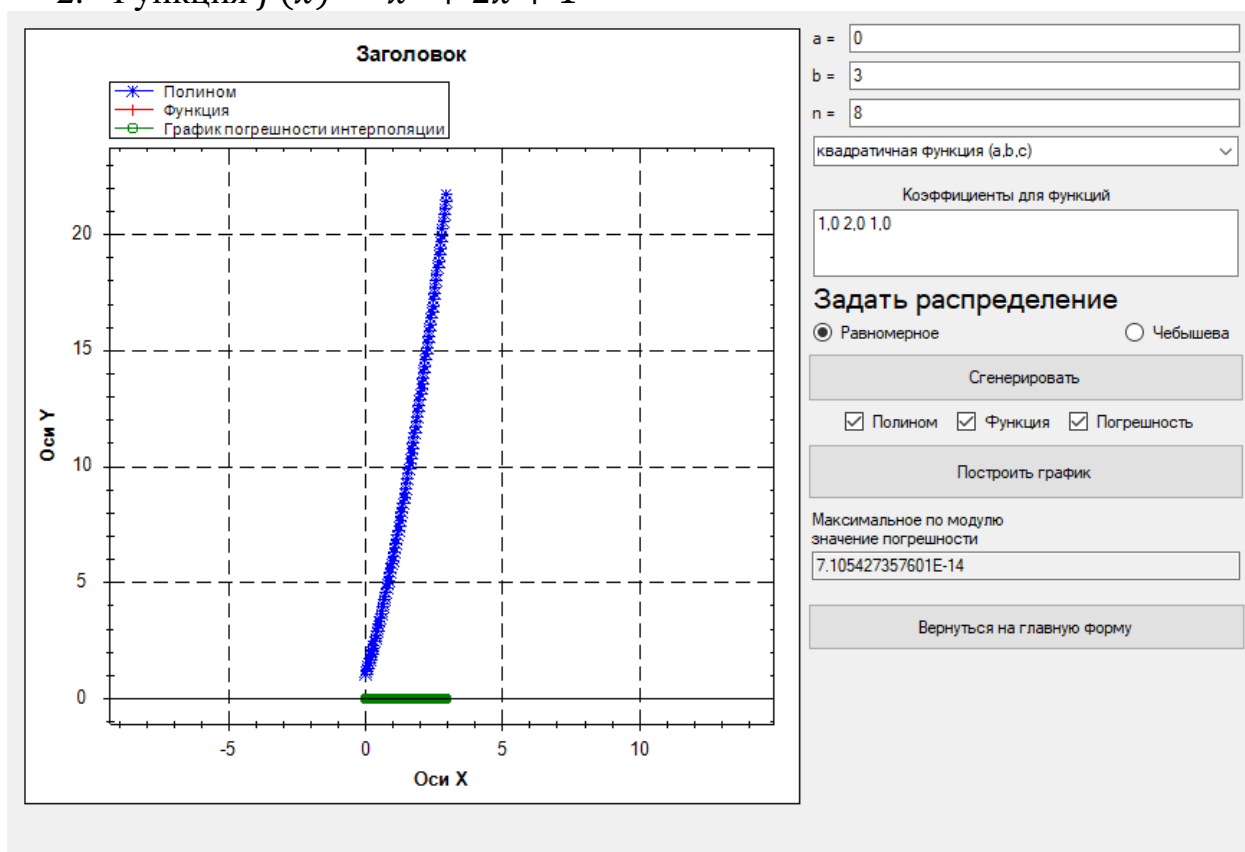


Рисунок 5 Функция $f(x) = x^2 + 2x + 1$

Значения функции Y:

Индекс	Значение
[0]	1
[1]	2.640625
[2]	4.5625
[3]	6.765625
[4]	9.25
[5]	12.015625
[6]	15.0625
[7]	18.390625

Значения полинома P:

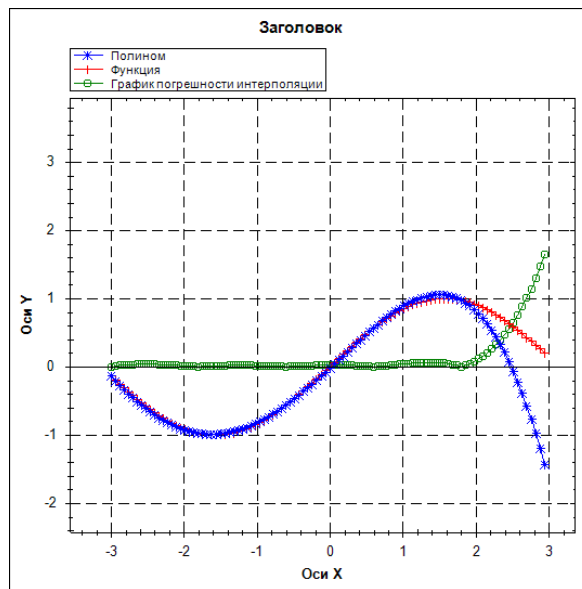
Индекс	Значение
[0]	1
[1]	2.640625
[2]	4.5624999999999991
[3]	6.7656249999999982
[4]	9.2499999999999947
[5]	12.0156249999999982
[6]	15.0624999999999954
[7]	18.3906249999999929

- Исследовать сходимость интерполяционного полинома $P_n(x)$ к интерполируемой функции $f(x)$ при увеличении n :

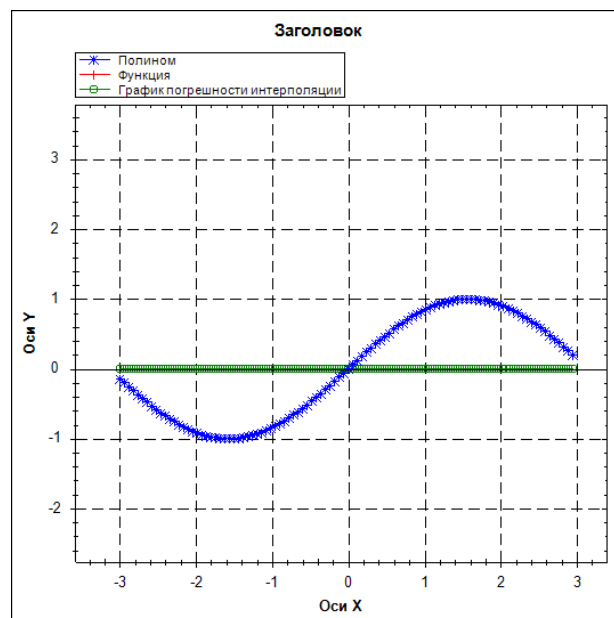
1. Привести пример функции $f(x)$, для которой $P_n(x)$ сходится к $f(x)$ при увеличении n ;

1.1. Функция $f(x) = \sin(x)$ на отрезке $[-5;5]$

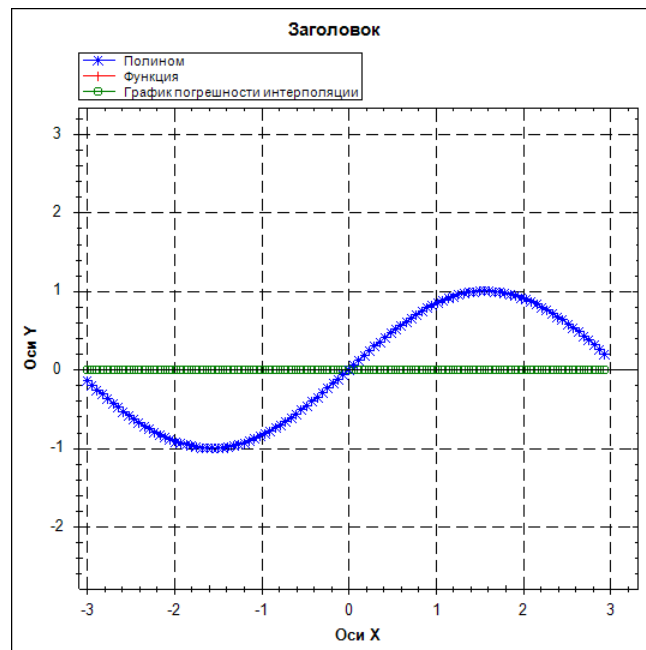
$N = 5$



$N = 10$



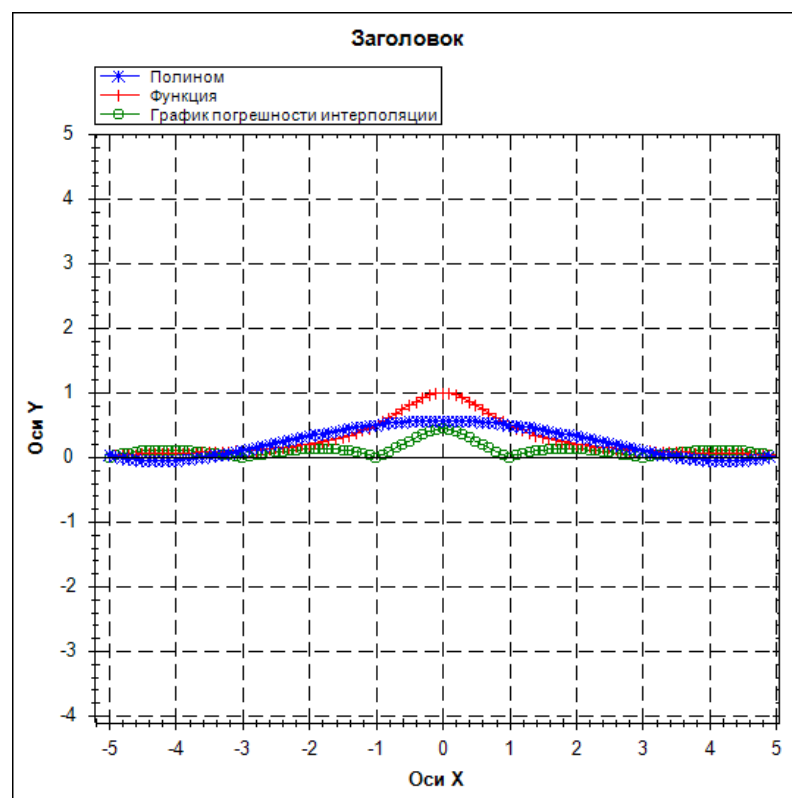
$N = 20$



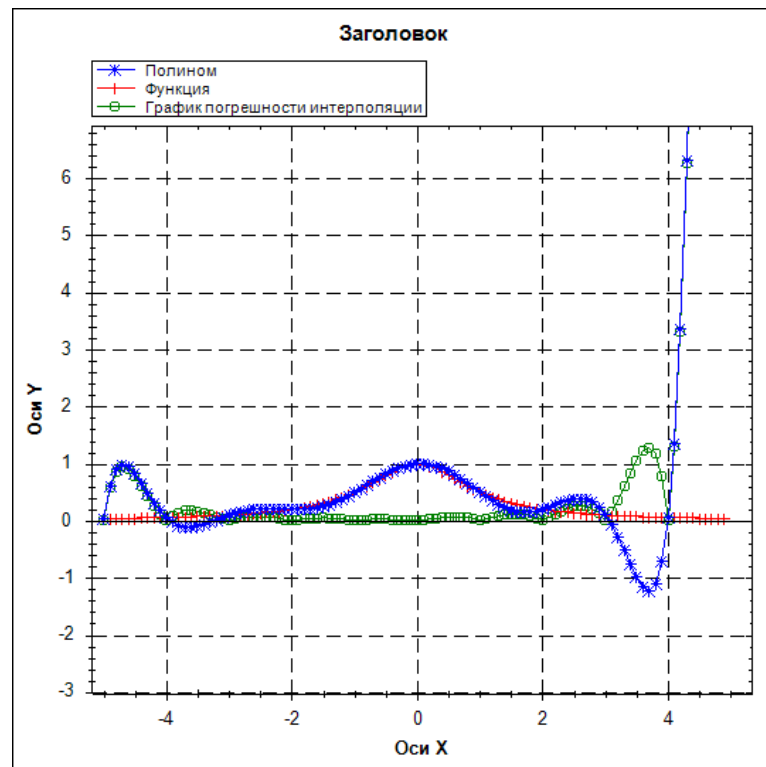
2. Привести пример функции $f(x)$, для которой $P_n(x)$ не сходится к $f(x)$ при увеличении n .

2.1 Пример Рунге на отрезке $[-5; 5]$

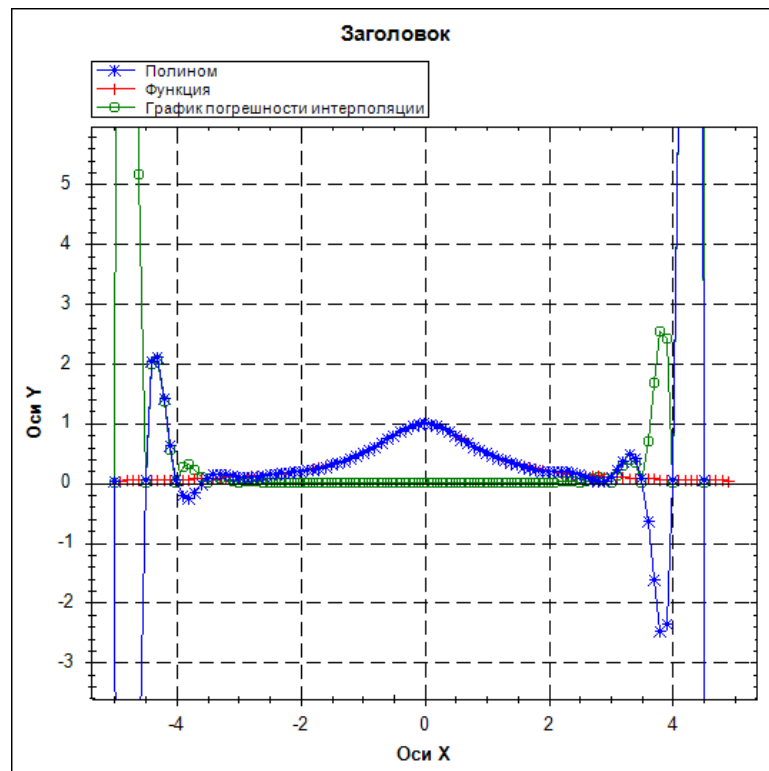
$N = 5$



$N = 10$



$N = 20$



Вывод

- 1) Вычислительный эксперимент подтверждает то, что построен действительно интерполяционный полином степени n ;
- 2) Вычислительный эксперимент действительно демонстрирует пример сходимости интерполяционного полинома к интерполируемой функции при увеличении степени полинома;
- 3) Вычислительный эксперимент действительно демонстрирует пример расходимости интерполяционного полинома при увеличении степени полинома;
- 4) Вычислительный эксперимент демонстрирует влияние выбора узлов интерполяции на точность полученного результата;
- 5) Вычислительный эксперимент демонстрирует влияние ошибок округления на точность полученного результата.

Вторая задача. Аппроксимация табличной функции (метод наименьших квадратов)

Постановка задачи

Реализовать аппроксимацию заданной табличной функции.

Входные данные:

- $f(x)$ — экспериментальная функция, где $x \in [a; b]$
- n — количество узлов на сетке

В моём случае аппроксимирующая функция $f(x) = a + \frac{b}{x}$

Теоретические сведения

Пусть в узлах x_0, x_1, \dots, x_n заданы значения функции $y_i = f(x_i)$, и надо построить приближающий эту функцию многочлен $Q_m(x)$, причём степень многочлена m меньше узлов. За меру отклонения многочлена

$$Q_m(x) = a + \frac{b}{x}$$

от функции $f(x)$ на множестве точек x_0, x_1, \dots, x_n принимают величину

$$S = \sum_{i=0}^n (Q_m(x) - y_i)^2$$

Очевидно, что S есть положительно определенная квадратичная функция коэффициентов a, b . Эти коэффициенты надо подбирать так, чтобы значение S было минимальным. Приравнявая частные производные по a, b нулю, получим систему $(m + 1)$ уравнений с $(m + 1)$ неизвестными a, b :

$$\frac{1}{2} \frac{\delta S}{\delta a} = \sum_{i=0}^n \left(a + \frac{b}{x} - y_i \right) * 1 = 0$$

$$\frac{1}{2} \frac{\delta S}{\delta b} = - \sum_{i=0}^n \left(a + \frac{b}{x} - y_i \right) * \frac{1}{x_i^2} = 0$$

Аппроксимация заданной табличной функции сводится к решению следующей системы уравнений:

$$\begin{bmatrix} s_0 & s_1 \\ s_1 & s_2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} t_0 \\ t_1 \end{Bmatrix}$$

Где (для функции $f(x) = a + \frac{b}{x}$)

$$s_0 = n + 1, s_1 = \sum_{i=0}^n \frac{1}{x_i}, s_2 = \sum_{i=0}^n \frac{1}{x_i^2}, t_0 = \sum_{i=0}^n y_i, t_1 = \sum_{i=0}^n \frac{y_i}{x_i}$$

Решение же данной системы можно выписать в следующем виде:

$$a_0 = \frac{t_0 s_2 - s_1 t_1}{s_0 s_2 - s_1^2}, \quad a_1 = \frac{t_1 s_0 - s_1 t_0}{s_0 s_2 - s_1^2}$$

```
private void CoefficientsCalculation()
{
    const int n = 2;

    var S = new double[n, n];
    var t = new double[n];

    for (var index = 0; index < N; index++)
    {
        var temp:double = 1 / XDoubles[index];
        S[0, 1] += temp;
        S[1, 1] += temp * temp;
        t[0] += FTableDoubles[index];
        t[1] += FTableDoubles[index] * temp;
    }

    S[0, 0] = N + 1;
    S[1, 0] = S[0, 1];

    ApproximationCoefficients = new double[2];
    ApproximationCoefficients[0] = (t[0] * S[1, 1] - S[0, 1] * t[1]) / (S[0, 0] * S[1, 1] - S[0, 1] * S[0, 1]);
    ApproximationCoefficients[1] = (t[1] * S[0, 0] - t[0] * S[0, 1]) / (S[0, 0] * S[1, 1] - S[0, 1] * S[0, 1]);
}
```

Рисунок 6 Реализация функции подсчёта коэффициентов

Вычислительный эксперимент

Проведём вычислительный эксперимент, чтобы проверить работоспособность полученного алгоритма:

При проведении экспериментов используется функция с коэффициентами a, b и количеством узлов n .

1. $a = 1.0, b = 1.0, n = 100$. Погрешность = $3.02185380057531\text{E-}05$

Отрезок $[a; b] = [-100; -1]$

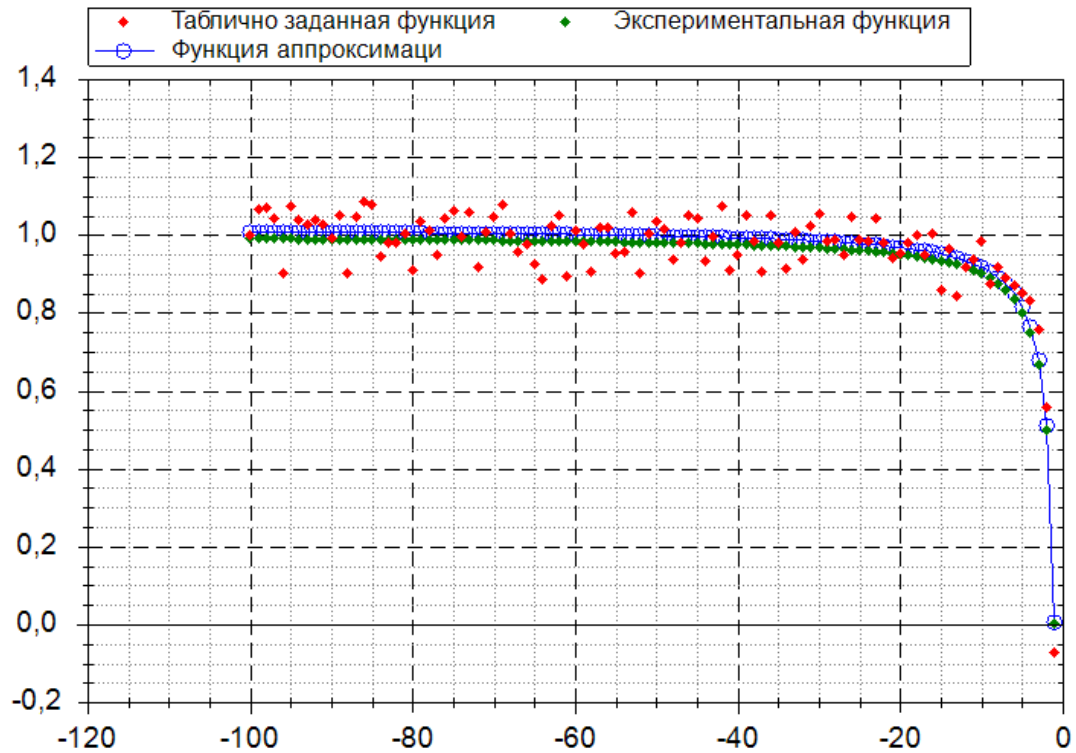


Рисунок 7 Результат эксперимента

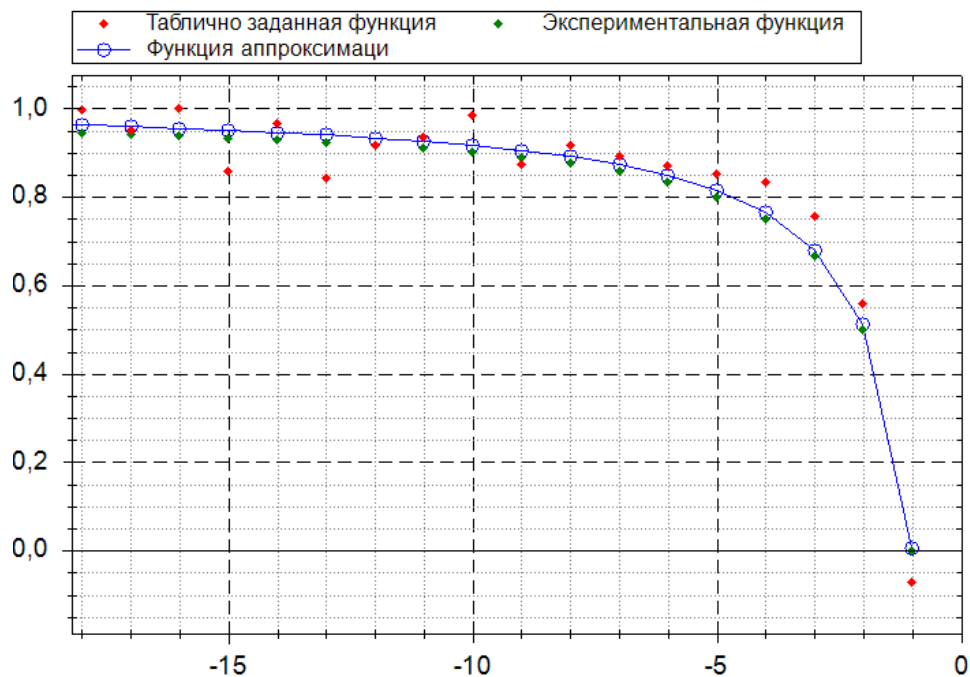
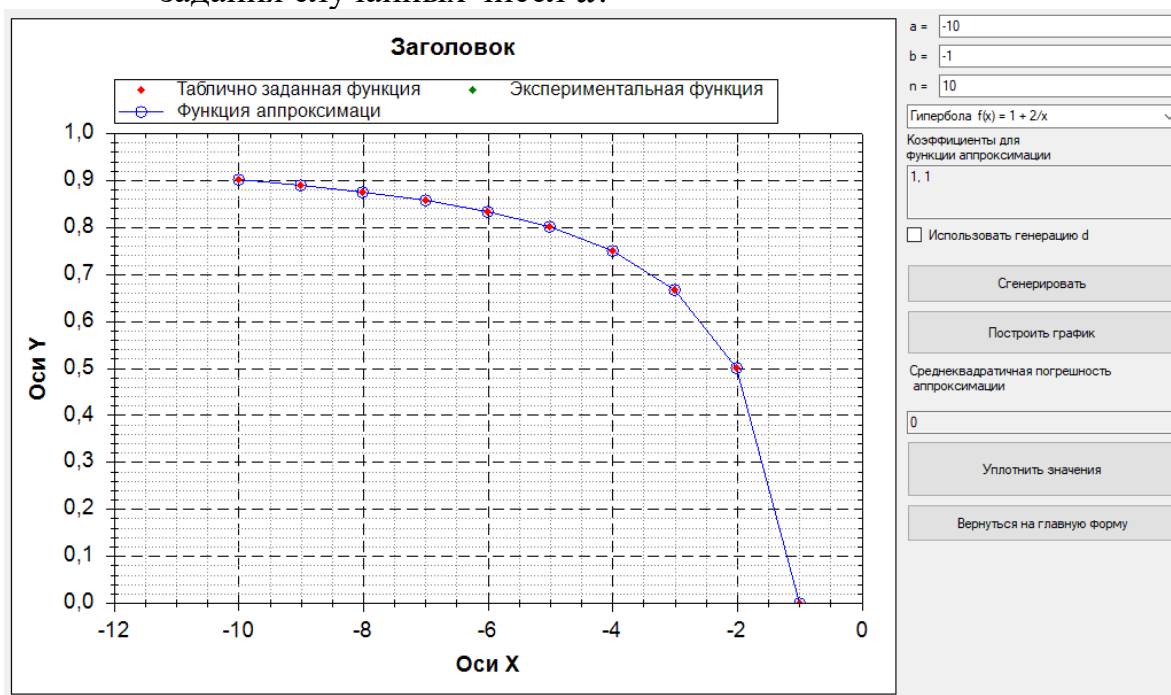


Рисунок 8 Приближенные графики

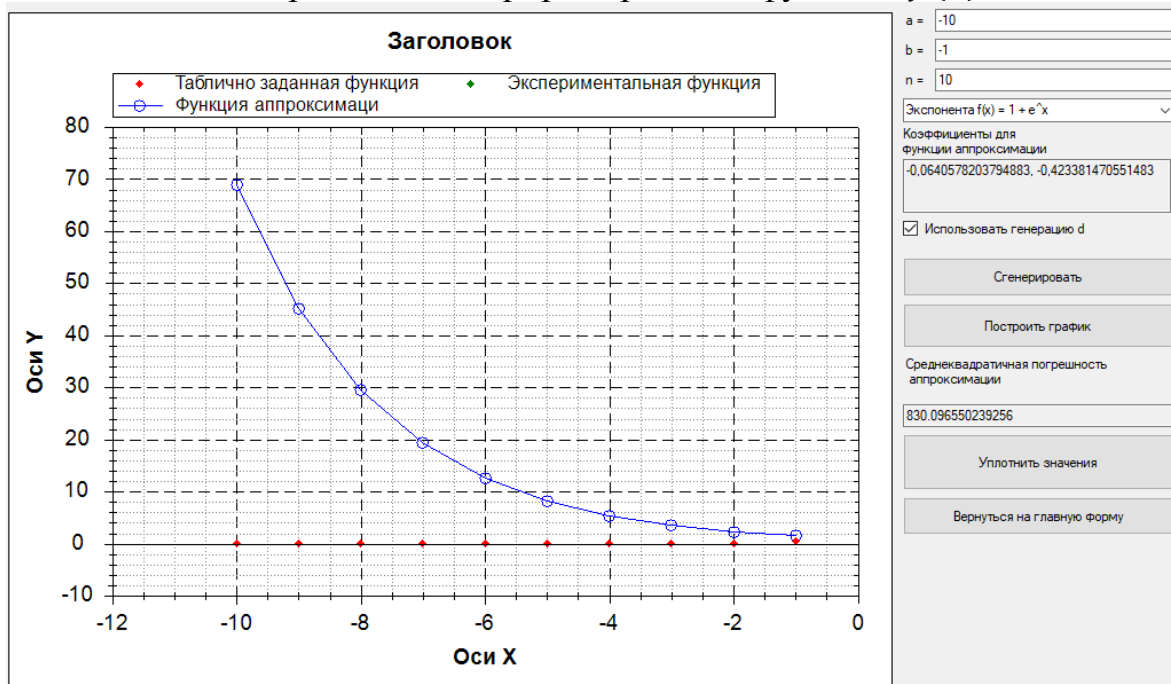
2. Проведём уплотнение функции $f(x) = 1 + \frac{1}{x}$, начиная с $n = 10$ на отрезке $[-10; -1]$

Значение n	Результат работы программы
10	<div data-bbox="534 309 1380 907"> </div> <div data-bbox="715 907 1197 1187"> <p>Коэффициенты для функции аппроксимации</p> <p>1,05317094698035, 1,11489259688514</p> <p>Среднеквадратичная погрешность аппроксимации</p> <p>0.00129430781234391</p> </div>
80	<div data-bbox="534 1209 1380 1792"> </div> <div data-bbox="715 1792 1197 2049"> <p>Коэффициенты для функции аппроксимации</p> <p>0,995682492592376, 0,975900189680578</p> <p>Среднеквадратичная погрешность аппроксимации</p> <p>2.4204727036996E-05</p> </div>

3. Рассмотрим также аппроксимирование табличной функции без задания случайных чисел d :



4. Также приведём контрпример в виде функции $f(x) = 1 + e^x$:



Вывод

После проделанной мною работы можно сделать вывод что вычислительный эксперимент демонстрирует зависимость между погрешностью аппроксимации и видом аппроксимирующей функции.

Третья задача. Интерполирование функций кубическими сплайнами.

Постановка задачи

Задан вид однопараметрической экспериментальной функции $f(x)$, значения которой определены на наборе узлов $\{x_i: a \leq x_i \leq b\}, i = 0, 1, \dots, n - 1$. Таким образом, имеет n штук узлов, в которых вычисляется значение экспериментальной функции. (При этом возможно получение сетки узлов как равномерным разбиением отрезка, т.е. $x_{i+1} = x_i + h, h = \text{const}$, так и разбиением Чебышева). Также задаются следующие краевые условия для интерполяционного сплайна: $S'''(x_0) = A, S'(x_n) = B$.

Требуется реализовать процедуру построения интерполяционного кубического сплайна $S(x)$ для таблично заданной функции $f(x)$, построенной по $n+1$ узлу интерполяции $x_i, i = 0, \dots, n$, а также для заданных краевых условий. Для построения кубического сплайна необходимо выразить $S(x)$ через коэффициенты c_i , получить систему для вычисления соответствующих коэффициентов и решить её методом прогонки.

Выписать коэффициенты в общем виде

Теоретические сведения

Сплайном, соответствующим данной функции $f(x)$ и данным узлам x_0, x_1, \dots, x_n , называется функция $S(x)$, удовлетворяющая следующим условиям:

- 1) На каждом сегменте $[x_i; x_{i+1}], i = 0, 1, \dots, n - 1$, функция $S(x)$ является многочленом третьей степени;
- 2) Функция $S(x)$, а также ее первая и вторая производные непрерывны на $[a; b]$;
- 3) $S(x_i) = f(x_i), i = 0, \dots, n$

Последнее условие называется условием интерполирования, а сплайн, определяемый условиями 1) – 3), называется также интерполяционным кубическим сплайном. На каждом из отрезков $[x_i; x_{i+1}], i = 0, 1, \dots, n - 1$, будем искать функцию $S(x) = S_i(x)$ в виде многочлена третьей степени:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

Наша задача – найти коэффициенты сплайновой кривой $S(x)$.

Рассм. многоч. 1-3:

$$S_i(x_i) = y_i; \quad S_i(x_{i+1}) = y_{i+1}$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \quad i = 0 \div (n-2)$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \quad \text{нуж. упр.}$$

Составим кривую:

$$S_i(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$$

$$S_i'(x) = b_i + 2c_i(x-x_i) + 3d_i(x-x_i)^2$$

$$S_i''(x) = 2c_i + 6d_i(x-x_i)$$

Обозначим за промежуток $h_i = x_{i+1} - x_i, \quad i = 0 \div (n-1)$

Из условия $S_i(x_i) = y_i \quad (i = 0 \div (n-1))$ получаем $a_i = y_i$

Из условия:

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \quad i = 0 \div (n-1) \quad \text{т.е.}$$

$$S_i'(x_{i+1}) = b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_i + 2c_i h_i + 3d_i h_i^2$$

$$S_{i+1}'(x_{i+1}) = b_{i+1} + 2c_{i+1}(x_{i+1} - x_{i+1}) + 3d_{i+1}(x_{i+1} - x_{i+1})^2 = b_{i+1}$$

$$\text{След. } b_i + 2h_i c_i + 3h_i^2 d_i - b_{i+1} = 0 \quad (*)$$

Из условия $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \quad (i = 0 \div n-2)$ получим:

$$2c_i + 6h_i d_i - 2c_{i+1} = 0 \Rightarrow d_i = \frac{2c_{i+1} - 2c_i}{3h_i}$$

Уже из $S_i(x_{i+1}) = y_{i+1}$ получим $a_i + b_i h_i + h_i^2 c_i + h_i^3 d_i = y_{i+1}, \quad i = 0 \div n-1$

или же: $y_i + h_i b_i + h_i^2 c_i + h_i^3 d_i = y_{i+1} \quad (\text{при } y_i = a_i)$

Подставим в упр. ~~выражение~~ b_i найденное для d_i через c_i , получим:

$$b_i = \frac{y_{i+1} - y_i}{h_i} - h_i c_i - \frac{1}{3} h_i (c_{i+1} - c_i)$$

Подставим в предыдущее соотношение b_i найденное, получим:

$$h_i c_i + 2(h_i + h_{i+1}) c_{i+1} + h_{i+1} c_{i+2} = 3 \left[\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i} \right]$$

где $i = 0 \div (n-2)$

Таким образом получено $n-1$ уравнение по c_i .
 $n+1$ неизвестных c_i . Кроме того, составим 2 граничных ур-я
 (из ур-я $S'''(x_0) = A; S''(x_n) = B$)

$$1) S'''(x_0) = A \Rightarrow \left(\text{т.к. } S'''(x_i) = d_i \right) d_0 = A$$

$$\text{т.к. } d_i = \frac{h_i}{3}(c_{i+1} - c_i) \quad (\text{ф-ла была получена ранее})$$

$$\text{то } c_0 - c_1 = -\frac{3A}{h_0}$$

$$2) S''(x_n) = B \Rightarrow h_{n-1} = B \Rightarrow h_{n-1}c_{n-1} + 2h_{n-1}c_n =$$

$$= 3 \left[B - \frac{y_n - y_{n-1}}{h_{n-1}} \right]$$

Получим оконч. б-у системы для нахождения c_i :

$$\begin{cases} c_0 - c_1 = -\frac{3A}{h_0} \end{cases}$$

$$\begin{cases} h_i c_i + 2(h_i + h_{i+1})c_{i+1} + h_{i+1}c_{i+2} = 3 \left[\frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_i} \right] \\ i = 0 \div (n-2) \end{cases}$$

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3 \left[B - \frac{y_n - y_{n-1}}{h_{n-1}} \right]$$

Полученная же относительно коэффициентов c_i система является трёхдиагональной, и для её решения можно воспользоваться методом прогонки (Томаса) решения СЛАУ.

Таким образом получим метод, который будет выполнять формирование векторов под трехдиагональную матрицу и решение данной матрицы методом прогонки

```
private void CalculateSpline()
{
    // Нижняя диагональ
    var bottomDiagonal = new double[N];
    // Главная диагональ
    var mainDiagonal = new double[N + 1];
    // Верхняя диагональ
    var topDiagonal = new double[N];
    // Правая часть
    var rightPart = new double[N + 1];
    // Результирующий вектор
    var resultVector = new double[N + 1];
    for (var i = 0; i < N; i++)
        bottomDiagonal[i] = topDiagonal[i] = HDoubles[i];
    bottomDiagonal[0] = -1;
    mainDiagonal[0] = 1;
    mainDiagonal[N] = 2 * HDoubles[N - 1];
    for (var i = 1; i < N; i++)
        mainDiagonal[i] = 2 * (HDoubles[i - 1] + HDoubles[i]);
    //левое условие
    rightPart[0] = 3 * HDoubles[0] * LeftCondition;
    //правое условие
    rightPart[N] =
        6 * (RightCondition - (YDoubles[N] - YDoubles[N - 1]) / HDoubles[N - 1]);
    for (var i = 1; i < N; i++)
    {
        rightPart[i] =
            6 * ((YDoubles[i + 1] - YDoubles[i]) / HDoubles[i] -
                (YDoubles[i] - YDoubles[i - 1]) / HDoubles[i - 1]);
    }

    LinearSystem.ThomasAlgorithm(
        bottomDiagonal,
        mainDiagonal,
        topDiagonal,
        ref resultVector,
        rightPart,
        N + 1
    );
    for (var i = 0; i < N; i++)
    {
        ADoubles[i] = YDoubles[i];
        CDoubles[i] = resultVector[i] / 2;
        BDoubles[i] =
            (YDoubles[i + 1] - YDoubles[i]) / HDoubles[i]
            - HDoubles[i] * CDoubles[i]
            - HDoubles[i] * (resultVector[i + 1] - resultVector[i]) / 6;
        DDoubles[i] =
            (resultVector[i + 1] - resultVector[i]) / (6 * HDoubles[i]);
    }
}
```

Вычислительный эксперимент

Исследуем зависимость погрешности интерполяции от количества подотрезков разбиения на примере нескольких функций:

Заданная функция	Количество подотрезков разбиения	Погрешность интерполяции при равномерном разбиении отрезка $[a; b]$	Погрешность интерполяции при разбиении Чебышева отрезка $[a; b]$
$f(x) = 3x^3 - 2x^2 + x + 16$ $x \in [-2; 2]$	10	2,88883938992512E-08	2,95813151751645E-07
	100	0	8,01833266450558E-10
$f(x) = 2\cos(x) - \cos(x)$ $x \in [0; \pi]$	10	4,50109061134185E-10	5,88304960302821E-11
	100	1,90315101677996E-15	6,18562090437536E-16
$f(x) = \cos\left(\frac{\pi x}{2}\right) + e^{-\frac{1}{x}}$ $x \in [0; \pi]$	10	3,83574202977854E-06	9,83574202977854E-06
	100	0,0017274698001	0,00107872599741

Графики исходной функции $f(x)$ и интерполяционного кубического сплайна $S(x)$, а также погрешности интерполяции:

$$1. f(x) = 3x^3 - 2x^2 + x + 16, x \in [-2; 2], n = 100$$

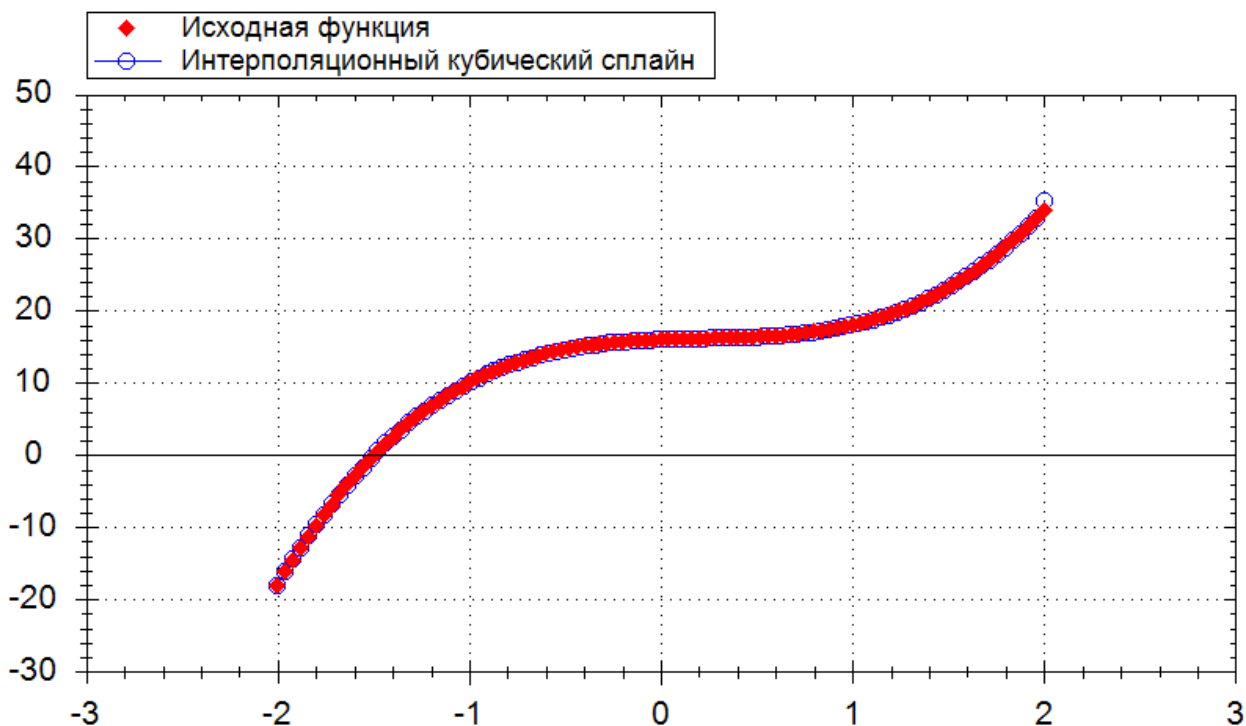


Рисунок 9 Графики исходной функции и кубического сплайна

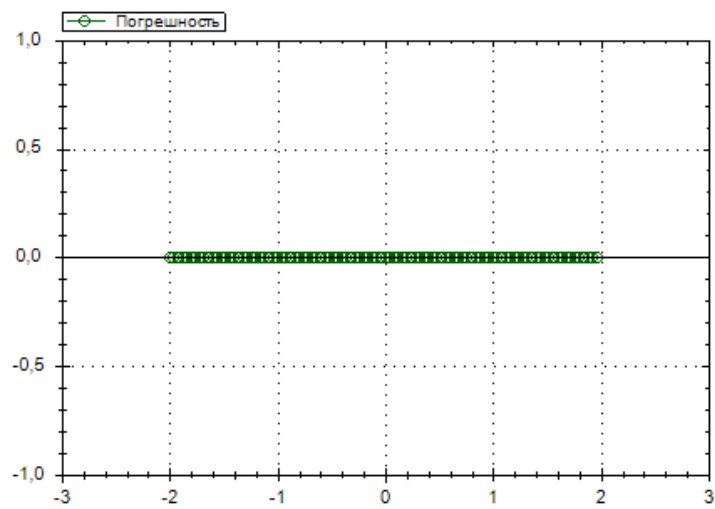


Рисунок 10 График погрешности

2. $f(x) = 2\cos(x) - \cos(x)$, $x \in [0; \pi]$, $n = 100$

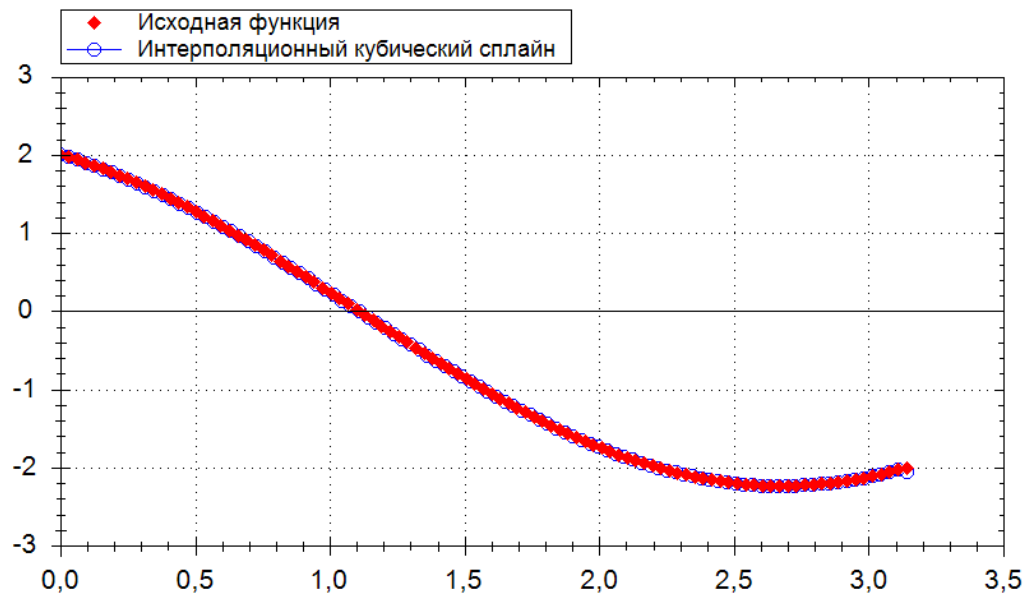


Рисунок 11 Графики исходной функции и интерполяционного кубического сплайна

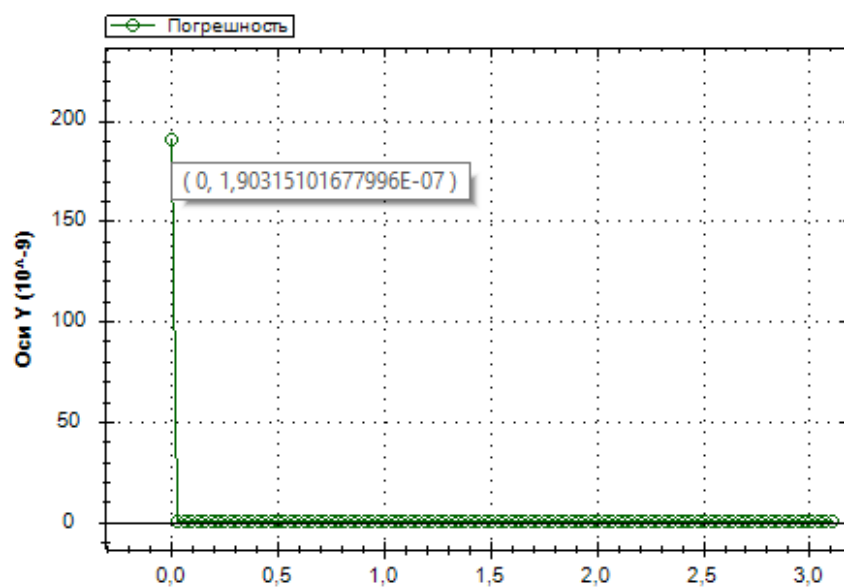














Рисунок 12 График погрешности

Вывод

1. Эксперимент действительно подтверждает, что строится кубический сплайн. Для демонстрации этого возьмем функцию $f(x) = x^2$ на отрезке $[0; 2]$ для 2 подотрезков при ГУ: $S'(x) = 0$, $S'''(x) = 4$ и построим для нее кубический сплайн:

В результате работы программы получим следующие коэффициенты

(a_i b_i c_i d_i соответственно)

 ADoubles {double[2]}	 BDoubles {double[2]}	 CDoubles {double[2]}	 DDoubles {double[2]}
 ADoubles[0] 0	 BDoubles[0] 0	 CDoubles[0] 1	 DDoubles[0] 0
 ADoubles[1] 1	 BDoubles[1] 1.9999999999	 CDoubles[1] 1.0000000000000002	 DDoubles[1] -7.4014868308343765E-17

Округлив полученные коэффициенты составим соответствующие сплайны:

$$S_0 = 0 + 0(x - 0) + 0(x - 0)^2 + 0(x - 0)^3 = x^2, x \in [0; 1]$$

$$S_1 = 1 + 2(x - 1) + (x - 1)^2 + 0(x - 1)^3 = 1 + 2x - 2 + x^2 - 2x + 1 + 0 = x^2, x \in [1; 2]$$

Ч.т.д.

2. Для первых двух функций из таблицы (см. «Вычислительный эксперимент») вычислительный эксперимент действительно демонстрирует пример сходимости интерполяционного кубического сплайна к интерполируемой функции при увеличении количества подотрезков разбиения.
3. Кроме того, вычислительный эксперимент демонстрирует пример расходимости интерполяционного кубического сплайна при увеличении количества подотрезков разбиения (для третьей функции из таблицы «Вычислительный эксперимент»).
4. Также в ходе вычислительного эксперимента была показана сходимость интерполяционного кубического сплайна при увеличении количества подотрезков разбиения в том случае, когда интерполяционный полином демонстрирует расходимость.
5. И, наконец, вычислительный эксперимент продемонстрировал влияние ошибок округления на точность полученного результата (см. п.1, «Выводы»).