ЛАБОРАТОРНАЯ РАБОТА №1.
ВАРИАНТ 6

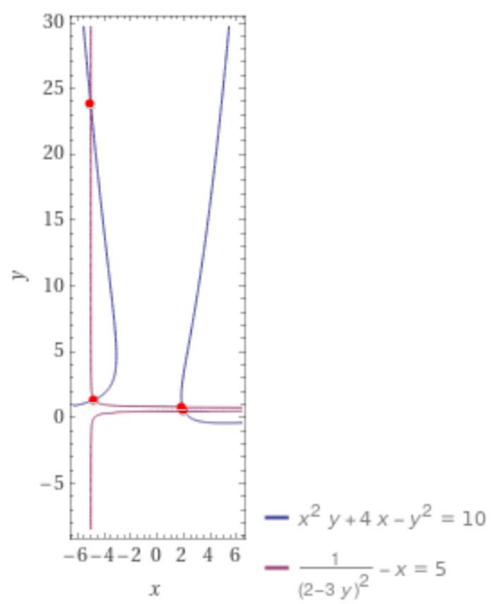
Выполнил:
студент 3 курса 13 группы кафедры ТП.

Петров Андрей Александрович

Задание №1: Необходимо найти решение системы нелинейных уравнений $f(x)= 0$ с точностью $\varepsilon = 10^{-6}$ с помощью метода Ньютона, и дискретного варианта метода Ньютона (подбор шагов в замене производной выполнить самостоятельно, обосновать свой выбор). Требуется предварительно выбрать начальное приближение (можно графически). Для каждого метода укажите количество итераций, необходимое для достижения заданной точности (полученное на практике), и значение $f(x^n)$, где x^n – полученное решение.
Сравните используемые методы решения систем нелинейных уравнений. Объясните быструю / медленную сходимость каждого из методов.

$$\begin{cases} x^2y - y^2 + 4x = 10; \\ \frac{1}{(3y-2)^2} - x = 5; \end{cases}$$

Ход работы:



Рассмотрев график, можно заметить, что корней всего 4. За начальное приближение возьмем (x0, y0) = (-6, 3);

$$f(x) = \begin{pmatrix} x^2y - y^2 + 4x - 10; \\ \frac{1}{(3y-2)^2} - x - 5; \end{pmatrix}$$

Построим матрицу Якоби:

$$J(x^k) = \begin{pmatrix} 2xy+4 & -1 \\ x^2-2y & -\frac{6}{(3y-2)^3} \end{pmatrix}$$

Обобщенный метод Ньютона на случай системы

$$x_{k+1} = x^k - f'(x^k)^{-1} f(x^k)$$

где $f' = \frac{df}{dx}$ – матрица Якоби

Реализовывать программу будем на языке C# с использованием библиотеки MathNet.Numerics.LinearAlgebra.

Реализуем заданные функции:

```
private static Vector<double> F(Vector<double> X)
{
    Vector<double> f = DenseVector.Create(X.Count, 0);
    f[0] = Math.Pow(X[0], 2) * X[1] - Math.Pow(X[1], 2) + 4 * X[0] - 10;
    f[1] = 1 / Math.Pow(3 * X[1] - 2, 2) - X[0] - 5;
    return f;
}
```

Реализуем матрицу Якоби:

```
private static Matrix<double> dF(Vector<double> X)
{
    Matrix<double> dF = DenseMatrix.Create(X.Count, X.Count, 0);
    dF[0, 0] = 2 * X[0] * X[1] + 4;
    dF[0, 1] = Math.Pow(X[0], 2) - 2 * X[1];
    dF[1, 0] = -1;
    dF[1, 1] = -6 / Math.Pow(3 * X[1] - 2, 3);

    return dF;
}
```

Реализация метода Ньютона будет выглядеть таким образом:

```
public static void Run(Vector<double> X, double epsilon)
{
    Vector<double> dX;
    Vector<double> XLast;
    Matrix<double> W;

    Console.WriteLine($"X0 = [{string.Join("; ", X)}]");

    var Dx = double.MaxValue;
    var iter = 0;

    while (Dx > epsilon)
    {
        iter++;
```

```
        XLast = X;
        W = dF(XLast);
        X = XLast - W.Inverse() * F(XLast);
        dX = X - XLast;
        Dx = dX.SumMagnitudes();
    }

    Console.WriteLine($"Count iteration = {iter}");
    Console.WriteLine($"X = [{string.Join("; ", X)}]");
    Console.WriteLine($"F(x) = [{string.Join("; ", F(X))}]");
}
```

Результат выполнения программы:

```
=================Newton Method==================
X0 = [-6; 3]
Count iteration = 6
X = [-4,768755187566936; 1,3598414348220473]
F(x) = [3,552713678800501E-15; 0]
================================================
```

Количество итераций для метода Ньютона: 6.

$x^n = (-4.767552, 1.359841)^T$

$f(x^n) = (0,0)^T$

Найдем решение системы нелинейных уравнений с помощь дискретного метода Ньютона.

Матрица Якоби изменится и будет выглядеть следующим образом

$$J(x^k) = \begin{pmatrix} \frac{f_1(x_1+h,x_2)-f_1(x_1,x_2)}{h} & \frac{f_1(x_1,x_2+h)-f_1(x_1,x_2)}{h} \\ \frac{f_2(x_1+h,x_2)-f_2(x_1,x_2)}{h} & \frac{f_2(x_1,x_2+h)-f_2(x_1,x_2)}{h} \end{pmatrix}$$

Возьмем шаг $h = 10^{-5}$

Реализуем матрицу Якоби для дискретного метода Ньютона:

```
private static Matrix<double> dF(Vector<double> X, double h)
{
    Matrix<double> dF = DenseMatrix.Create(X.Count, X.Count, 0);
    dF[0, 0] = (F1(X[0] + h, X[1]) - F1(X[0], X[1])) / h;
    dF[0, 1] = (F1(X[0], X[1] + h) - F1(X[0], X[1])) / h;
    dF[1, 0] = (F2(X[0] + h, X[1]) - F2(X[0], X[1])) / h;
    dF[1, 1] = (F2(X[0], X[1] + h) - F2(X[0], X[1])) / h;
    return dF;
}
```

Реализация дискретного метода Ньютона будет выглядеть таким образом:

```
public static void Run(Vector<double> X, double h, double epsilon)
{
    Vector<double> dX;
    Vector<double> XLast;
    Matrix<double> W;

    Console.WriteLine($"X0 = [{string.Join("; ", X)}]");
    Console.WriteLine($"h = {h}");

    var Dx = double.MaxValue;
    var iter = 0;

    while (Dx > epsilon)
    {
        iter++;
        XLast = X;
        W = dF(XLast, h);
        X = XLast - W.Inverse() * F(XLast);
        dX = X - XLast;
        Dx = dX.SumMagnitudes();
    }

    Console.WriteLine($"Count iteration = {iter}");
    Console.WriteLine($"X = [{string.Join("; ", X)}]");
    Console.WriteLine($"F(x) = [{string.Join("; ", F(X))}]");
}
```

Результат выполнения программы:

```
=============Discrete Newton Method=============
X0 = [-6; 3]
h = 1E-05
Count iteration = 6
X = [-4,768755187566936; 1,3598414348220473]
F(x) = [3,552713678800501E-15; 0]
================================================
```

Исходя из результатов выполнения обоих методов видно, что у них получились одинаковые результаты. Причиной этого послужил достаточно малая величина шага. Однако если мы возьмем большую величину шага, скорость ходимости изменится в худшую сторону, либо метод вовсе перестанет сходиться.

Вот, например, результат выполнения программы дискретного метода Ньютона с увеличенным шагом $h = 1$:

```
============Discrete Newton Method============
X0 = [-6; 3]
h = 1
Count iteration = 11
X = [-4,768755158295182; 1,3598414351390824]
F(x) = [-2,56205574089563E-07; -2,948328070573325E-08]
=============================================
```

Как видно, количество итераций увеличилось почти в 2 раза.

Задание №2: Рассмотрим функцию

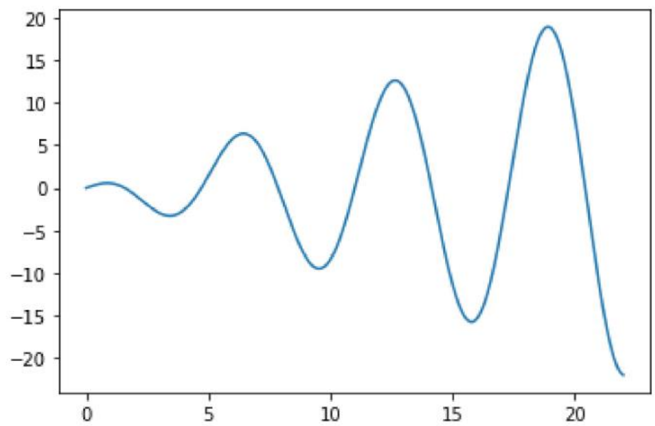
$$f(x) = xcosx, x \in [0, 22];$$

1. Построить интерполяционные многочлены функции $f(x)$ по 6, 12, 18 равноотстоящим узлам.
2. Построить интерполяционные многочлены функции $f(x)$ по 6, 12, 18 узлам Чебышева.
3. Построить интерполяционные сплайны третьего порядка функции $f(x)$ по 6, 12, 18 равноотстоящим узлам.
4. На графике функции $f(x)$ выбрать 100 случайных точек на отрезке и построить по ним наилучшие среднеквадратичные приближения для базиса $\varphi_i(x) = x^i, i = \overline{0, n}$ при $n = 1, 2, 4, 6$.
5. Вывести отчет в формате .txt. В отчет должно входить:

- Время, затраченное на построение каждого интерполяционного многочлена.
- Время, затраченное на построение каждого сплайна.
- Время, затраченное на построение каждого среднеквадратичного приближения.

Ход работы:

График заданной функции:



1. Построим интерполяционные многочлены функции f(x) по 6, 12, 18 равносторонним узлам. Для этого используем барицентрическую интерполяционную формулу.

$$P_n(x) = \frac{\sum_{i=0}^{n} y_i \frac{v_i}{x - x_i}}{\sum_{i=0}^{n} \frac{v_i}{x - x_i}},$$

где весовой коэффициент $v_i = \frac{1}{\omega'_{n+1}(x_i)} = \frac{1}{\prod_{j \neq i}(x_i - x_j)}, \quad i = \overline{0, n}.$

Реализуем функцию нахождения весового коэффициента:

```
private static double WeightFactor(Vector<double> X, int i)
{
    double multiply = 1;
    for (var j = 0; j < X.Count; j++)
        if (i != j) multiply *= (X[i] - X[j]);
    return 1 / multiply;
}
```

Реализуем функцию нахождения $P_n(x)$:

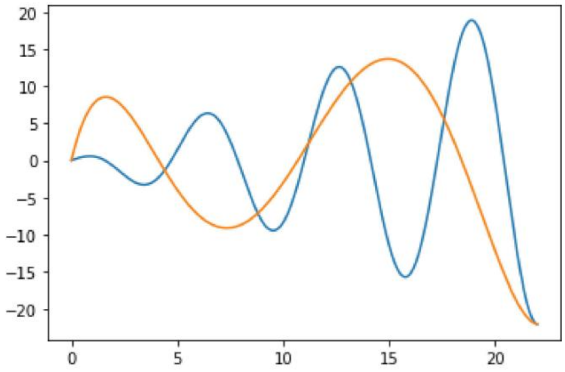
```
private static double Pn(double X0, Vector<double> X, Vector<double> Y, Func<double,
double> F)
{
    if (X.Exists(x => x.Equals(X0))) return F(X0);

    double numerator = 0;
    double denominator = 0;
    for (var i = 0; i < X.Count; i++)
    {
        var v = WeightFactor(X, i);
        numerator += Y[i] * v / (X0 - X[i]);
        denominator += v / (X0 - X[i]);
    }
    return numerator / denominator;
}
```

Программа:

```
X0 = DenseVector.OfArray(new double[] { 0, 22 });
var xArr = Generate.LinearRange(X0[0], 0.02, X0[1]);
...
Vector<double> X = DenseVector.OfArray(Generate.LinearSpaced(nodes, X0[0], X0[1]));
Vector<double> Y = Barycentric.Y(F, X);
Vector<double> P = DenseVector.Build.Dense(xArr.Length, 0);

time.Start();
for (var i = 0; i < xArr.Length; i++)
    P[i] = Pn(xArr[i], X, Y, F);
time.Stop();
...
```

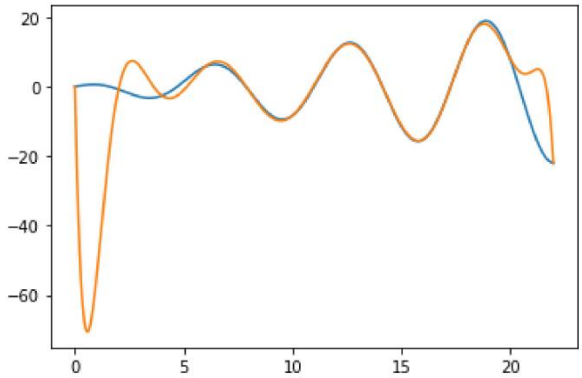
Построим интерполяционный многочлен функции f(x) по 6 равносторонним узлам и засечем время:

```
======Barycentric Equally Spaced Nodes(6)=======
X0 = [0; 22]
arrX = file: results/6.txt, row 1; Number of items: 1101
P = file: results/6.txt, row 2; Number of items: 1101
X = file: results/6.txt, row 3; Number of items: 6
Y = file: results/6.txt, row 4; Number of items: 6
Time = 4,3824ms
================================================
```



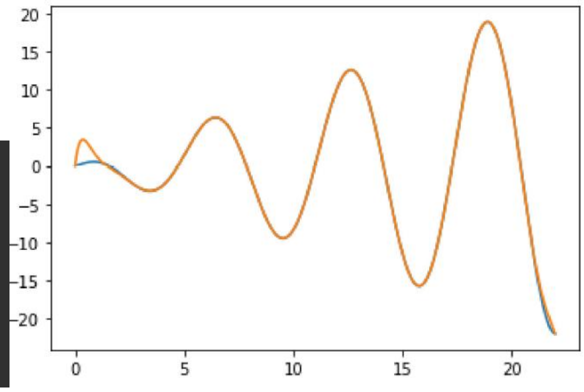
Построим интерполяционный многочлен функции f(x) по 12 равносторонним узлам и засечем время:

```
======Barycentric Equally Spaced Nodes(12)======
X0 = [0; 22]
arrX = file: results/12.txt, row 1; Number of items: 1101
P = file: results/12.txt, row 2; Number of items: 1101
X = file: results/12.txt, row 3; Number of items: 12
Y = file: results/12.txt, row 4; Number of items: 12
Time = 7.6435ms
================================================
```



Построим интерполяционный многочлен функции f(x) по 18 равносторонним узлам и засечем время:

```
======Barycentric Equally Spaced Nodes(18)======
X0 = [0; 22]
arrX = file: results/18.txt, row 1; Number of items: 1101
P = file: results/18.txt, row 2; Number of items: 1101
X = file: results/18.txt, row 3; Number of items: 18
Y = file: results/18.txt, row 4; Number of items: 18
Time = 13.9709ms
================================================
```



2. Построим интерполяционные многочлены функции f(x) по 6, 12, 18 узлам Чебышева. Для этого используем барицентрическую интерполяционную формулу.

Использование чебышевских узлов интерполяции, помимо прочего, существенно повышает эффективность барицентрической интерполяционной формулы. Таком образом применив чебышевские узлы к формуле весовых коэффициентов, она принимает более простой вид:

$$v_i = (-1)^i \sin \frac{2i+1}{2n+2}\pi.$$

Реализуем измененную функцию нахождения весового коэффициента барицентрической интерполяционной формулы:

```
private static double WeightFactor(int i, double degree)
{
    return Math.Pow(-1, i) * Math.Sin((2 * i + 1) / (2 * degree + 2) * Math.PI);
}
```

Корни многочлена Чебышева, масштабированного на заданный промежуток (оптимальный узлы интерполяции на отрезке) будем находить с помощью формулы:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos \frac{\pi(2i+1)}{2n+2}, \quad i = \overline{0, n}.$$

Реализуем функция нахождения корней многочлена:

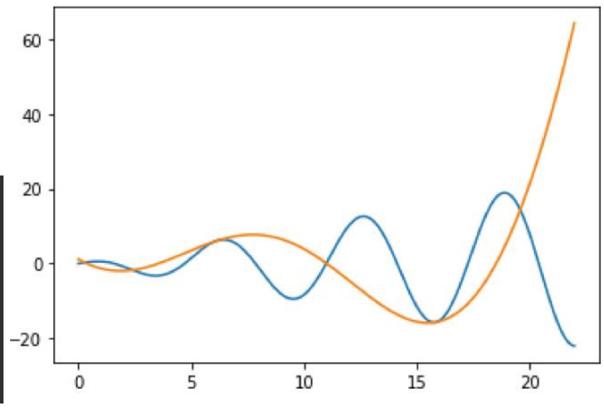
```
private static double Xn(double a, double b, double i, double n)
{
    return (a + b) / 2 + ((a - b) / 2) * Math.Cos(Math.PI * (2 * i + 1) /
                                                         (2 * n + 2));
}
```

Измененная программа

```
...
Vector<double> X = DenseVector.Build.Dense(nodes, 0);
Vector<double> Y = DenseVector.Build.Dense(nodes, 0);
Vector<double> P = DenseVector.Build.Dense(xArr.Length, 0);

for (var i = 0; i < nodes; i++)
{
    X[i] = Xn(X0[0], X0[1], i, nodes);
    Y[i] = F(X[i]);
}

time.Start();
for (var i = 0; i < xArr.Length; i++)
    P[i] = Pn(xArr[i], X, Y, F, nodes);
time.Stop();

...
```

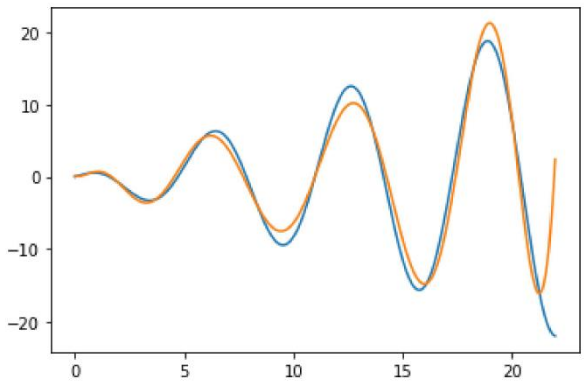
Построим интерполяционный многочлен функции f(x) по 6 узлам Чебышева и засечем время:

```
==ChebyshevBarycentric Equally Spaced Nodes(6)==
X0 = [0; 22]
arrX = file: results/6.txt, row 1; Number of items: 1101
P = file: results/6.txt, row 2; Number of items: 1101
X = file: results/6.txt, row 3; Number of items: 6
Y = file: results/6.txt, row 4; Number of items: 6
Time = 1.3056ms
================================================
```



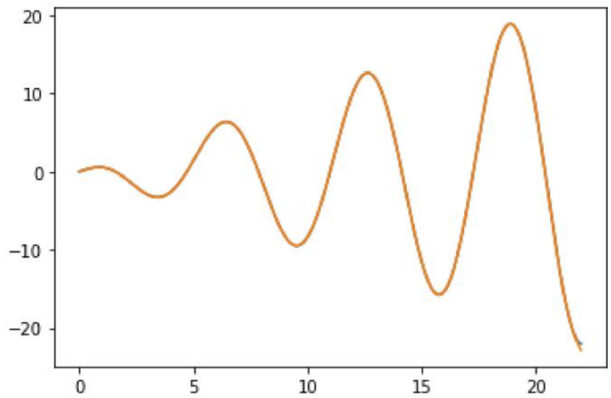
Построим интерполяционный многочлен функции f(x) по 12 узлам Чебышева и засечем время:

```
==ChebyshevBarycentric Equally Spaced Nodes(12)=
X0 = [0; 22]
arrX = file: results/12.txt, row 1; Number of items: 1101
P = file: results/12.txt, row 2; Number of items: 1101
X = file: results/12.txt, row 3; Number of items: 12
Y = file: results/12.txt, row 4; Number of items: 12
Time = 1.7992ms
================================================
```



Построим интерполяционный многочлен функции f(x) по 18 узлам Чебышева и засечем время:

```
==ChebyshevBarycentric Equally Spaced Nodes(18)=
X0 = [0; 22]
arrX = file: results/18.txt, row 1; Number of items: 1101
P = file: results/18.txt, row 2; Number of items: 1101
X = file: results/18.txt, row 3; Number of items: 18
Y = file: results/18.txt, row 4; Number of items: 18
Time = 1.6896ms
================================================
```



Рассмотрев графики, можно отметить, что при использовании чебышевских узлов приближение улучшается на краях промежутка, но становится немного хуже в середине. При этом на 18 узлах сходимость очень хорошая.

3. Построим интерполяционные сплайны третьего порядка функции $f(x)$ по 6, 12, 18 равноотстоящим узлам. Каждый «кусок» сплайна будем искать в виде:

$$s_i(x) = \alpha_i + \beta_i(x - x_i) + \frac{\gamma_i}{2}(x - x_i)^2 + \frac{\delta_i}{6}(x - x_i)^3, \quad x \in \Delta_i = [x_{i-1}, x_i].$$

В качестве граничных условий возьмем $S''(x_0) = S''(x_n)$, откуда следует $\gamma 0 = \gamma_n = 0$

Реализуем функцию нахождения $s_i(x)$

```
private static double Spline(double t, Vector<double> alpha, Vector<double> beta,
    Vector<double> gamma, Vector<double> delta, int i)
{
    return alpha[i] + beta[i] * t + gamma[i] * Math.Pow(t, 2) / 2 + delta[i] *
Math.Pow(t, 3) / 6;
}
```

Реализуем функции нахождения alpha, beta, gamma, delta

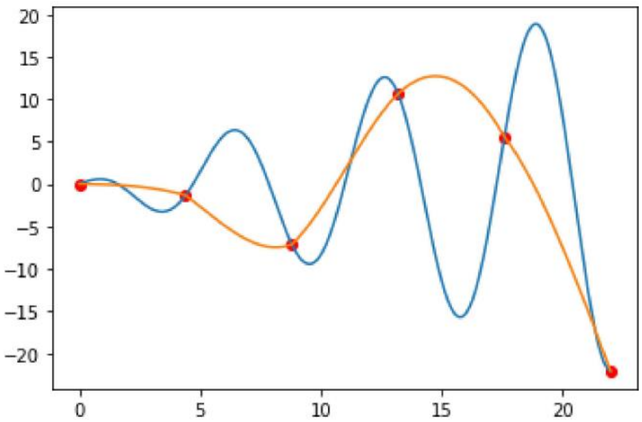
```
private static double GetDeltaByIndex(int i, double h, Vector<double> gamma) =>
    (gamma[i] - gamma[i - 1]) / h;

private static double GetBetaByIndex(int i, double h, Vector<double> alpha,
                                                        Vector<double> gamma) =>
    (alpha[i] - alpha[i - 1]) / h + h * (2 * gamma[i] + gamma[i - 1]) / 6;

private static Vector<double> GetGamma(double h, Vector<double> alpha)
{
    var n = alpha.Count;
    var cI = 1 / 2;
    var eI = cI;
    Vector<double> b = DenseVector.Build.Dense(n - 2, 0);

    for (var i = 1; i < n - 1; i++)
        b[i - 1] = 3 * (alpha[i + 1] + alpha[i - 1] - 2 * alpha[i]) / Math.Pow(h, 2);

    var matrix = DenseMatrix.Build.Dense(n - 2, n - 2);

    for (var i = 0; i < n - 2; i++) {
        var col = DenseVector.Build.Dense(n - 2, 0);
        col[i] = 2;

        if (i == 0) col[i + 1] = cI;
        else if (i == n - 3) col[i - 1] = eI;
        else {
            col[i - 1] = eI;
            col[i + 1] = cI;
        }

        matrix.SetColumn(i, col);
    }
    var gamma = matrix.Solve(b).ToList();
    gamma.Add(0);
    gamma.Insert(0, 0);
    return DenseVector.Build.DenseOfArray(gamma.ToArray());
}
```

Основная часть программы:

```
Vector<double> X = DenseVector.OfArray(Generate.LinearSpaced(nodes, range[0],
range[1]));
Vector<double> Y = GetY(F, X);
Vector<double> S = DenseVector.Build.Dense(xArr.Length, 0);
var h = (range[1] - range[0]) / (nodes - 1);

time.Start();
Vector<double> gamma = GetGamma(h, Y);
var delta = DenseVector.Build.Dense(gamma.Count, 0);
var beta = DenseVector.Build.Dense(gamma.Count, 0);

for (var i = 1; i < nodes; i++)
{
    delta[i] = GetDeltaByIndex(i, h, gamma);
    beta[i] = GetBetaByIndex(i, h, Y, gamma);
}

for (var i = 0; i < xArr.Length; i++)
{
    var index = 1;
    while (xArr[i] > X[index]) index++;

    S[i] = Spline(xArr[i] - X[index], Y, beta, gamma, delta, index);
}
time.Stop();
```

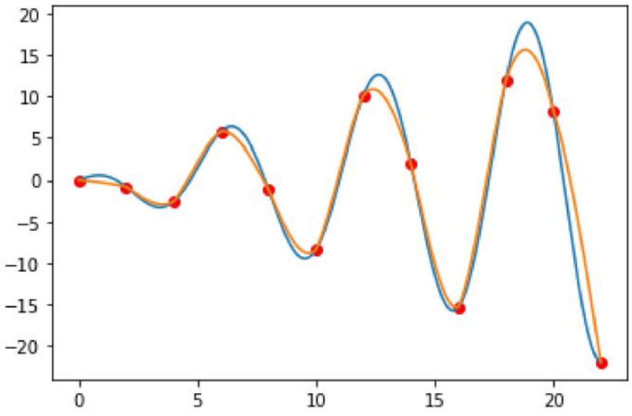
Построим интерполяционный сплайн третьего порядка функции $f(x)$ по 6 равноотстоящим узлам.

```
======CubicSpline Equally Spaced Nodes(6)=======
Range = [0; 22]
h = 4.4
arrX = file: results/6.txt, row 1; Number of items: 1101
S = file: results/6.txt, row 2; Number of items: 1101
X = file: results/6.txt, row 3; Number of items: 6
Y = file: results/6.txt, row 4; Number of items: 6
beta = file: results/6.txt, row 5; Number of items: 6
gamma = file: results/6.txt, row 6; Number of items: 6
delta = file: results/6.txt, row 7; Number of items: 6
Time = 8.3898ms
================================================
```



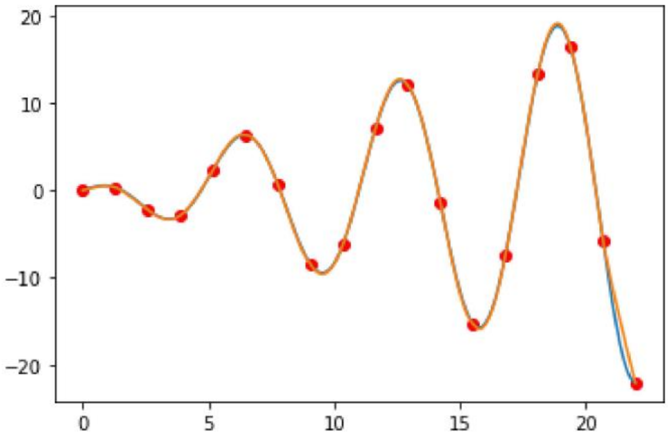
Построим интерполяционный сплайн третьего порядка функции $f(x)$ по 12 равноотстоящим узлам.

```
======CubicSpline Equally Spaced Nodes(12)======
Range = [0; 22]
h = 2
arrX = file: results/12.txt, row 1; Number of items: 1101
S = file: results/12.txt, row 2; Number of items: 1101
X = file: results/12.txt, row 3; Number of items: 12
Y = file: results/12.txt, row 4; Number of items: 12
beta = file: results/12.txt, row 5; Number of items: 12
gamma = file: results/12.txt, row 6; Number of items: 12
delta = file: results/12.txt, row 7; Number of items: 12
Time = 0.4976ms
================================================
```



Построим интерполяционный сплайн третьего порядка функции $f(x)$ по 18 равноотстоящим узлам.

```
======CubicSpline Equally Spaced Nodes(18)======
Range = [0; 22]
h = 1.2941176470588236
arrX = file: results/18.txt, row 1; Number of items: 1101
S = file: results/18.txt, row 2; Number of items: 1101
X = file: results/18.txt, row 3; Number of items: 18
Y = file: results/18.txt, row 4; Number of items: 18
beta = file: results/18.txt, row 5; Number of items: 18
gamma = file: results/18.txt, row 6; Number of items: 18
delta = file: results/18.txt, row 7; Number of items: 18
Time = 0.9049ms
================================================
```



4. На графике функции $f(x)$ выберем 100 случайных точек на отрезке и построим по ним наилучшие среднеквадратичные приближения для базиса $\boldsymbol{\varphi_i(x) = x^i, i = \overline{0, n}}$ при $\boldsymbol{n = 1, 2, 4, 6}$.

```
private static Vector<double> GetCoeffs(Func<double, double> F, Vector<double> X,
Vector<double> Y, int n)
{
    var B = DenseVector.Build.Dense(n + 1, 0);
    for (var i = 0; i < n + 1; i++) {
        var pair = X.Zip(Y, (x, y) => new { X = x, Y = y });
        B[i] = pair.Sum(p => p.Y * Math.Pow(p.X, i));
    }

    var matrix = DenseMatrix.Build.Dense(n + 1, n + 1, (i, j) => {
        return X.Sum(x => Math.Pow(x, j + i));
    });
    Console.WriteLine(matrix);
    return matrix.Solve(B);
}

private static double Compute(double x0, int n, Vector<double> coeffs)
{
```

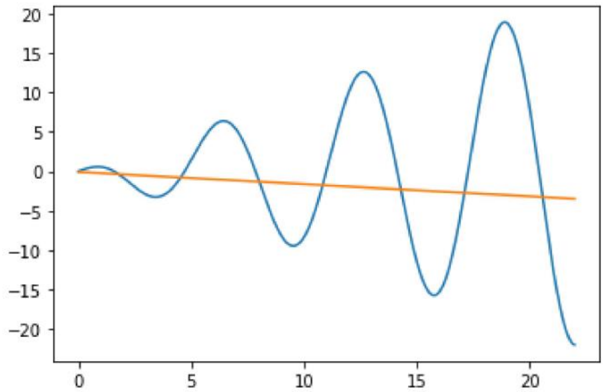
```
    var sum = 0.0;
    for (var i = 0; i < n + 1; i++)
        sum += coeffs[i] * Math.Pow(x0, i);
    return sum;
}
```

Основная программа:

```
time.Start();
Vector<double> X = DenseVector.Build.Random(100);
for (var i = 0; i < X.Count; i++)
    X[i] = rnd.NextDouble() * 22;
Vector<double> Y = GetY(F, X);
Vector<double> Rms = DenseVector.Build.Dense(xArr.Length, 0);

var coeffs = GetCoeffs(F, X, Y, degree);
for (var i = 0; i < xArr.Length; i++)
    Rms[i] = Compute(xArr[i], degree, coeffs);
time.Stop();
```

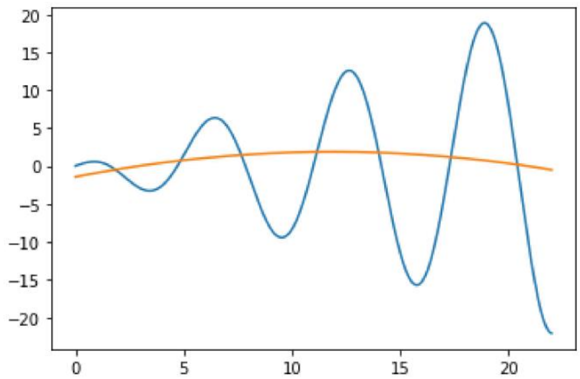
n=1

```
=============RmsApproximation(1)===============
Range = [0; 22]
Degree = 1
arrX = file: results/1.txt, row 1; Number of items: 1101
Rms = file: results/1.txt, row 2; Number of items: 1101
X = file: results/1.txt, row 3; Number of items: 100
Y = file: results/1.txt, row 4; Number of items: 100
Time = 12.1078ms
================================================
```



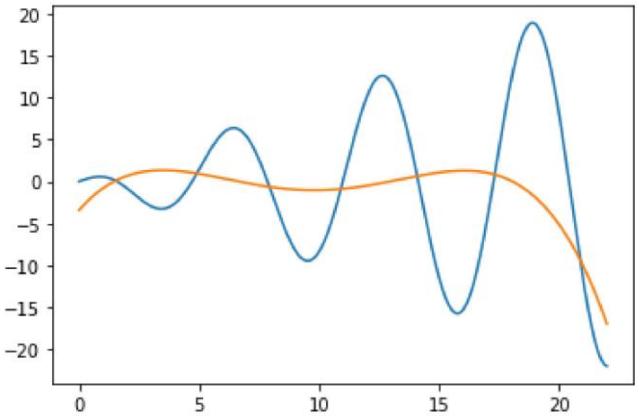
n=2

```
=============RmsApproximation(2)===============
Range = [0; 22]
Degree = 2
arrX = file: results/2.txt, row 1; Number of items: 1101
Rms = file: results/2.txt, row 2; Number of items: 1101
X = file: results/2.txt, row 3; Number of items: 100
Y = file: results/2.txt, row 4; Number of items: 100
Time = 0.5142ms
================================================
```



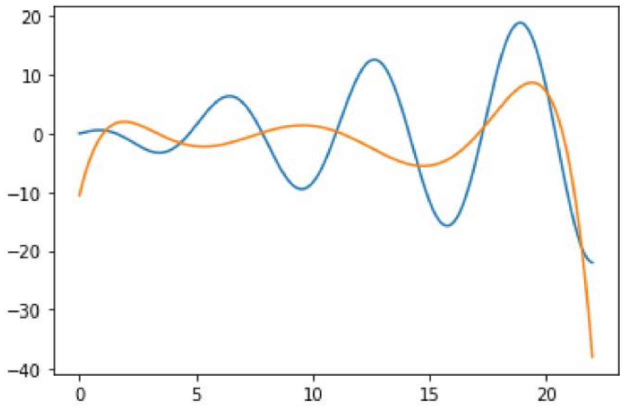
n=4

```
=============RmsApproximation(4)==============
Range = [0; 22]
Degree = 4
arrX = file: results/4.txt, row 1; Number of items: 1101
Rms = file: results/4.txt, row 2; Number of items: 1101
X = file: results/4.txt, row 3; Number of items: 100
Y = file: results/4.txt, row 4; Number of items: 100
Time = 1.0471ms
=============================================
```



n=6

```
=============RmsApproximation(6)==============
Range = [0; 22]
Degree = 6
arrX = file: results/6.txt, row 1; Number of items: 1101
Rms = file: results/6.txt, row 2; Number of items: 1101
X = file: results/6.txt, row 3; Number of items: 100
Y = file: results/6.txt, row 4; Number of items: 100
Time = 1.3973ms
=============================================
```



Задание №3: Рассмотрим функцию

$$g(x,y) = cosxy, x \in [0,4];\; y \in [0,4]$$

Для функции, соответствующей вашему варианту, проделать следующее:

1. Построить интерполяционные многочлены двух переменных функции $g(x, y)$ на прямоугольнике по сеткам 6×6, 12×12, 18×18 равноотстоящих узлов.

2. Построить бикубические сплайны функции $g(x, y)$ на прямоугольнике по сеткам 6×6, 12×12, 18×18 равноотстоящих узлов.

1. Построим интерполяционные многочлены двух переменных функции $g(x, y)$ на прямоугольнике по сеткам 6×6, 12×12, 18×18 равноотстоящих узлов.