

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ  
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №3  
З ДИСЦИПЛІНИ «Математичне моделювання та чисельні методи»  
НА ТЕМУ «Наближення функцій»

Виконав:  
студент гр. ПЗП-17-1  
Кириченко О. В.  
Перевірив:  
Матвеев Д. І.

Харків – 2020

**Мета роботи:** Навчитись використовувати математичне середовище MATLAB для апроксимації та інтерполяції. Відтворити досліджену поведінку математичного середовища за допомогою мови програмування.

Задля локанічності нижче наведено загальні методи на мові C#, які будуть використані у завданнях:

```
public Plot MakeInterpolationTaskPlot<T>(
    IEnumerable<Interpolator.Point> knownPoints, double delta
) where T : Interpolator, new() {
    IEnumerable<Interpolator.Point> interpolatedPoints =
        new T().Interpolate(knownPoints, delta);

    Plot plot = new Plot();
    plot.PlotSignalPoints(knownPoints, Color.Red, 5);
    plot.PlotSignalPoints(interpolatedPoints, Color.Blue);
    return plot;
}

public abstract class Interpolator
{
    public struct Point
    {
        public double X { get; set; }
        public double Y { get; set; }

        public Point(double x, double y)
        {
            X = x;
            Y = y;
        }
    }
}
```

```

    public abstract IEnumerable<Point> Interpolate(
        IEnumerable<Point> knownPoints, double? delta
    );
}

public class LinearInterpolator : Interpolator
{
    public override IEnumerable<Point> Interpolate(
        IEnumerable<Point> knownPoints, double? delta
    ) {
        double[] xs = delta.HasValue ? Range.Make(
            knownPoints.First().X, knownPoints.Last().X, delta.Value
        ) : knownPoints.Select(point => point.X).ToArray();
        double[] ys = xs.Select(
            x => Interpolate(knownPoints, x)
        ).ToArray();

        return xs.And(ys);
    }

    private double Interpolate(
        IEnumerable<Point> knownPoints, double x
    ) {
        if (x == knownPoints.Last().X)
            return knownPoints.Last().Y;
        if (x == knownPoints.First().X)
            return knownPoints.First().Y;

        List<Point> knownPointsList = knownPoints.ToList();

        int i = 0;
        for (; i < knownPointsList.Count - 1 && (knownPointsList[i].X
            > x || knownPointsList[i + 1].X < x); ++i) ;
    }
}

```

```

        return knownPointsList[i].Y +
            (knownPointsList[i + 1].Y - knownPointsList[i].Y) *
            (x - knownPointsList[i].X) /
            (knownPointsList[i + 1].X - knownPointsList[i].X);
    }
}

public class PolyInterpolator : Interpolator
{
    public override IEnumerable<Point> Interpolate(
        IEnumerable<Point> knownPoints, double? delta
    ) {
        int size = knownPoints.Count();

        double[,] w = Matrix.Make(
            size, size + 1,
            (i, j) => Math.Pow(knownPoints.ElementAt(i).X, j)
        );

        double[] y = knownPoints.Select(point => point.Y).ToArray();

        double[] a = Matrix.LinSolve(w, y);
        IEnumerable<Point> interpolated = null;
        if (!delta.HasValue)
        {
            double[] xs = knownPoints.Select(
                point => point.X
            ).ToArray();
            interpolated = xs.And(Poly.Val(a, xs));
        }
        else
        {
            double[] xs = Range.Make(

```

```

        knownPoints.First().X,
        knownPoints.Last().X,
        delta.Value
    );
    interpolated = xs.And(Poly.Val(a, xs));
}

return interpolated;
}
}

public static class Range
{
    public static double[] Make(double start, double end, double delta)
    {
        int count = (int)Math.Ceiling((end - start) / delta);
        return Enumerable.Range(0, count).Select(
            i => i * delta + start
        ).ToArray();
    }
}

public static class Extensions
{
    public static Point AddToForm(
        this Plot plot, Form form, Point location
    ) {
        Bitmap chart = plot.Render();

        PictureBox picture = new PictureBox();

        picture.Size = chart.Size;
        picture.Location = location;
        picture.BackgroundImage = chart;
    }
}

```

```

        form.Controls.Add(picture);

        return new Point(location.X, location.Y + chart.Height);
    }

    public static IEnumerable<Interpolator.Point> And(
        this double[] x, double[] y
    ) {
        return Enumerable.Range(0, x.Length).Select(
            i => new Interpolator.Point(x[i], y[i])
        );
    }

    public static void PlotSignalPoints(
        this Plot plot,
        IEnumerable<Interpolator.Point> points,
        Color? color = null, double thickness = 1
    ) {
        plot.PlotSignalXY(
            points.Select(point => point.X).ToArray(),
            points.Select(point => point.Y).ToArray(),
            color, thickness
        );
    }
}

public static class Matrix
{
    public static T[,] Make<T>(
        int height, int width, Func<int, int, T> generator
    ) {
        T[,] matrix = new T[height, width];

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

```

```

        for (int j = 0; j < width; ++j)
            matrix[i, j] = generator(i, j);

    return matrix;
}

public static (int height, int width) GetMatrixSize<T>(T[,] matrix)
{
    return (matrix.GetLength(0), matrix.GetLength(1));
}

public static double[,] ConvertToDouble(int[,] matrix)
{
    var size = GetMatrixSize(matrix);
    double[,] result = new double[size.height, size.width];
    for (int i = 0; i < size.height; ++i)
        for (int j = 0; j < size.width; ++j)
            result[i, j] = (double)matrix[i, j];
    return result;
}

public static double[] ConvertToDouble(int[] vector)
{
    return vector.Select(item => (double)item).ToArray();
}

public static T[,] VectorToMatrix<T>(T[] vector, bool inline)
{
    if (inline)
    {
        T[,] result = new T[1, vector.Length];
        for (int i = 0; i < vector.Length; ++i)
            result[0, i] = vector[i];
        return result;
    }
    else

```

```

        {
            T[,] result = new T[vector.Length, 1];
            for (int i = 0; i < vector.Length; ++i)
                result[i, 0] = vector[i];
            return result;
        }
    }

    public static double[,] Multiply(double[,] a, double[,] b)
    {
        var sizea = GetMatrixSize(a);
        var sizeb = GetMatrixSize(b);

        if (sizea.width != sizeb.height)
            throw new ArgumentException();

        double[,] result = new double[sizea.height, sizeb.width];

        for (int i = 0; i < sizea.height; ++i)
            for (int j = 0; j < sizeb.width; ++j)
                for (int k = 0; k < sizea.width; ++k)
                    result[i, j] += a[i, k] * b[k, j];

        return result;
    }

    public static T[] MatrixToVector<T>(T[,] matrix)
    {
        var size = GetMatrixSize(matrix);

        if (size.height == 1)
        {
            T[] vector = new T[size.width];
            for (int i = 0; i < size.width; ++i)

```



```

        vector[i] = matrix[0, i];
    }
    return vector;
}

else if (size.width == 1)
{
    T[] vector = new T[size.height];
    for (int i = 0; i < size.height; ++i)
        vector[i] = matrix[i, 0];
    return vector;
}

throw new ArgumentException();
}

public static T[,] GetMinor<T>(T[,] matrix, int row, int column)
{
    var size = GetMatrixSize(matrix);
    T[,] submatrix = new T[size.height - 1, size.width - 1];

    int di = 0;
    for (int i = 0; i < size.height; ++i)
    {
        if (i == row)
        {
            di = -1;
            continue;
        }

        int dj = 0;
        for (int j = 0; j < size.width; ++j)
        {
            if (j == column)
            {

```

```

        dj = -1;
        continue;
    }

    submatrix[i + di, j + dj] = matrix[i, j];
}
}
return submatrix;
}

public static double Det(double[,] matrix)
{
    var size = GetMatrixSize(matrix);

    if (size.height != size.width)
        throw new ArgumentException();

    if (size.width == 2 && size.height == 2)
        return matrix[0, 0] * matrix[1, 1] - matrix[0, 1] *
            matrix[1, 0];

    return Enumerable.Range(0, size.width).Sum(
        i => Math.Pow(-1, i) * matrix[0, i] *
            Det(GetMinor(matrix, 0, i))
    );
}

public static T[,] Transpose<T>(T[,] matrix)
{
    var size = GetMatrixSize(matrix);
    T[,] transposed = new T[size.width, size.height];
    for (int i = 0; i < size.height; ++i)
        for (int j = 0; j < size.width; ++j)
            transposed[j, i] = matrix[i, j];
}

```

```

        return transposed;
    }

    public static void Multiply(double[,] matrix, double value)
    {
        var size = GetMatrixSize(matrix);
        for (int i = 0; i < size.height; ++i)
            for (int j = 0; j < size.width; ++j)
                matrix[i, j] *= value;
    }

    public static double[,] Invert(double[,] matrix)
    {
        var size = GetMatrixSize(matrix);

        if (size.width != size.height)
            return PseudoInvert(matrix);

        double det = Det(matrix);

        if (det == 0)
            throw new ArgumentException();

        double[,] transposed = Transpose(matrix);
        double[,] complements = new double[size.height, size.width];

        for (int i = 0; i < size.height; ++i)
            for (int j = 0; j < size.width; ++j)
                complements[i, j] = Math.Pow(-1, i + j) *
                    Det(GetMinor(transposed, i, j));
        Multiply(complements, 1 / det);
        return complements;
    }

```

```

public static double[,] PseudoInvert(double[,] matrix)
{
    var size = GetMatrixSize(matrix);
    double[,] transposed = Transpose(matrix);

    if (size.height > size.width)
        return Multiply(
            Invert(Multiply(transposed, matrix)),
            Transposed
        );
    else
        return Multiply(
            transposed,
            Invert(Multiply(matrix, transposed))
        );
}

public static double[] LinSolve(double[,] x, double[] y)
{
    return MatrixToVector(
        Multiply(Invert(x), VectorToMatrix(y, false))
    );
}

}

public static class Poly
{
    public static double[] Fit(double[] x, double[] y, int k)
    {
        if (x.Length != y.Length)
            throw new ArgumentException();

        int n = x.Length;

```

```

        double[] powersums = new double[2 * k + 1];
        for (int i = 0; i <= 2 * k; ++i)
            powersums[i] = Enumerable.Range(0, n).Sum(
                j => Math.Pow(x[j], i)
            );

        double[, ] xm = new double[k + 1, k + 1];
        double[] ym = new double[k + 1];

        for (int i = 0; i <= k; ++i)
        {
            for (int j = 0; j <= k; ++j)
                xm[i, j] = powersums[i + j];
            ym[i] = Enumerable.Range(0, n).Sum(
                j => Math.Pow(x[j], i) * y[j]
            );
        }
        return Matrix.LinSolve(xm, ym);
    }

    public static Func<double, double> Get(double[] p)
    {
        return x => p.Select((a, i) => a * Math.Pow(x, i)).Sum();
    }

    public static double[] Val(double[] p, double[] x)
    {
        Func<double, double> poly = Get(p);
        return x.Select(xi => poly(xi)).ToArray();
    }
}

```

1. Створити вектори вимірювань (X, Y), згідно з наданих даних.

MATLAB	C#
<pre> a =      0.9000  &gt;&gt; X = [ -1.8 - 0.05 * sqrt(a); -1.2; -0.5; 1 + 0.5 * log10(a); 0.6 + 0.6 * a; 3.5 + 0.1 * a ]  X =     -1.8474    -1.2000    -0.5000     0.9771     1.1400     3.5900  &gt;&gt; Y = [ 1 - 0.12 * exp(a); 0; 0.5 + 0.35 * log(a); 1 - 0.2 * sqrt(a); 0.5 + 0.25 * cos(sqrt(a)); 1.2 - 0.65 * exp(sqrt(a)) ]  Y =      0.7048          0     0.4631     0.8103     0.6457    -0.4785 </pre>	<pre> double a = 0.9;  double[] x = new double[] {     -1.8 - 0.05 * Math.Sqrt(a),     -1.2,     -0.5,     1 + 0.5 * Math.Log10(a),     0.6 + 0.6 * a,     3.5 + 0.1 * a };  double[] y = new double[] {     1 - 0.12 * Math.Exp(a),     0,     0.5 + 0.35 * Math.Log(a),     1 - 0.2 * Math.Sqrt(a),     0.5 + 0.25 * Math.Cos(Math.Sqrt(a)),     1.2 - 0.65 * Math.Exp(Math.Sqrt(a)) }; </pre>

2. Для заданих значень векторів X та Y обчислити наближену функцію з використанням формул лінійної інтерполяції. Побудувати графіки, де x змінюється від X0 до X5 із кроком 0.02.

## MATLAB

```
function [XS, ys] = linearInterpolation(xp, yp, delta)
    XS = xp(1):delta:xp(length(xp));
    ys = zeros(size(XS));

    ys(1) = yp(1);
    ys(length(ys)) = yp(length(yp));

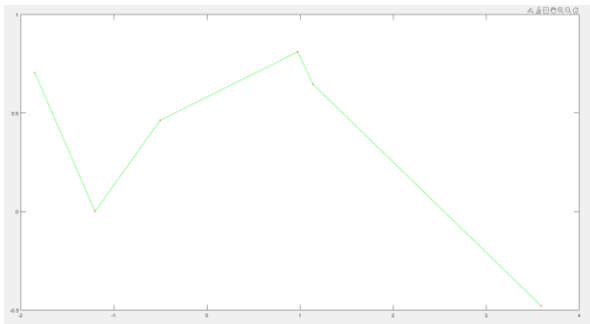
    for i = 2:(length(XS) - 1)
        ys(i) = linearInterpolationStep(xp, yp, XS(i));
    end

function [y] = linearInterpolationStep(xp, yp, x)
    i = findInterpolationIndex(xp, x);
    y = yp(i) + (yp(i + 1) - yp(i)) * (x - xp(i)) / (xp(i + 1) - xp(i));

function [i] = findInterpolationIndex(xp, x)
    for i = 1:(length(xp) - 1)
        if (xp(i) <= x && xp(i + 1) >= x)
            break;
        end
    end

>> [XS, YS] = linearInterpolation(X, Y, 0.02)

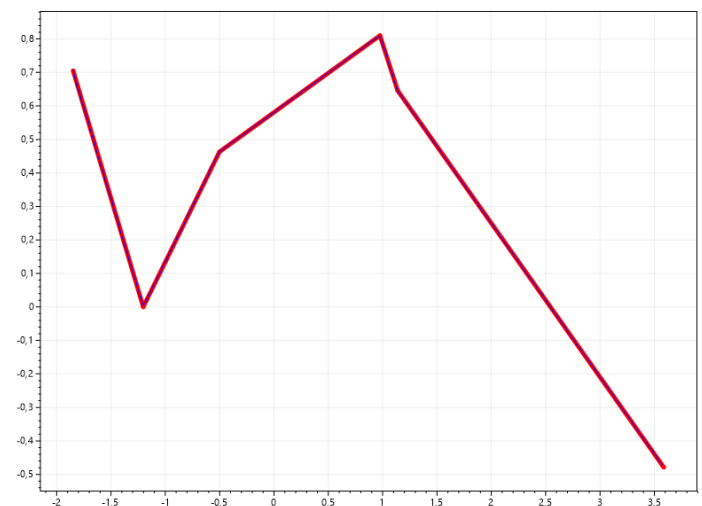
>> plot(X, Y, 'r', XS, YS, 'g')
```



## C#

```
IEnumerable<Interpolator.Point> knownPoints =
x.And(y);
Task1(form, new Point(0, 0), knownPoints, 0.02);

public Point Task1(
    Form form, Point location,
    IEnumerable<Interpolator.Point> knownPoints,
    double delta
) {
    Plot plot =
        MakeInterpolationTaskPlot<LinearInterpolator>(
            knownPoints, delta
        );
    return plot.AddToForm(form, location);
}
```



3. Для заданих значень векторів X та Y обчислити наближену функцію з використанням інтерполяційного алгебраїчного многочлена. Побудувати графіки, де x змінюється від X0 до X5 із кроком 0.01.

## MATLAB

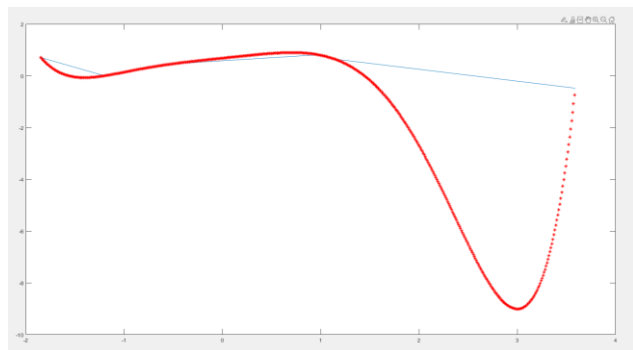
```
function [xs, ys] = polyInterpolation(x, y, delta)
    w = getW(x);
    as = linsolve(w, y);

    if(delta == 0)
        xs = x;
        ys = polyval(flip(as), x);
    else
        xs = x(1):delta:x(length(x));
        ys = polyval(flip(as), xs);
    end
end
```

```
function [w] = getW(x)
    len = length(x);
    w = zeros(len, len + 1);

    for i = 1:len
        for j = 1:(len + 1)
            w(i, j) = x(i) ^ (j - 1);
        end
    end
end
```

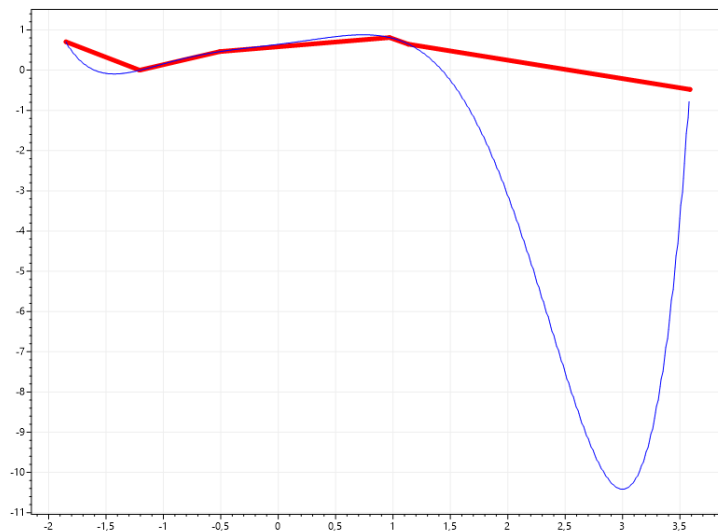
```
>> [XS YS] = polyInterpolation(X, Y, 0.01)
```



## C#

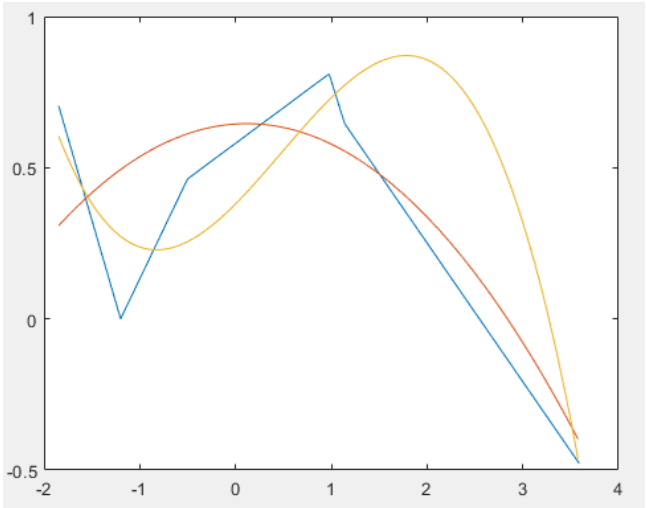
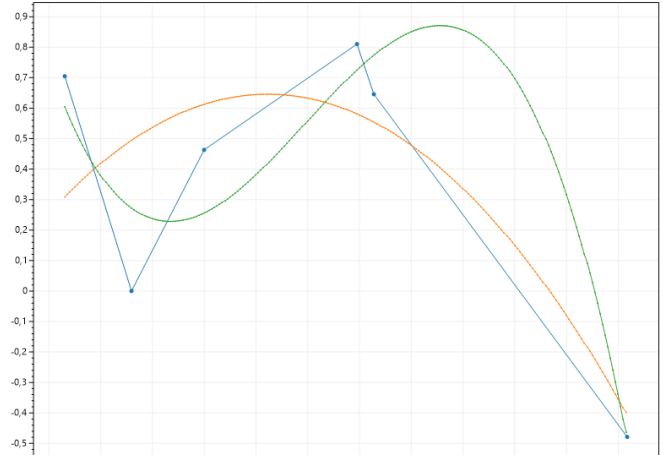
```
IEnumerable<Interpolator.Point> knownPoints =
    x.And(y);
Task2(form, point1, knownPoints, 0.01);

public Point Task2(
    Form form, Point location,
    IEnumerable<Interpolator.Point> knownPoints,
    double delta
) {
    Plot plot =
        MakeInterpolationTaskPlot<PolyInterpolator>(
            knownPoints, delta
        );
    plot.AddToForm(form, location);
}
```





4. Для заданих значень векторів X та Y вибрати коефіцієнти апроксимуючого поліному 2-го та 3-го ступенів. Побудувати графіки, де x змінюється від X0 до X5 із кроком 0.025.

MATLAB	C#
<pre data-bbox="105 447 755 798">&gt;&gt; XS = X(1):0.025:X(length(X)); &gt;&gt; AP2 = polyfit(X, Y, 2)   AP2 =     -0.0872    0.0207    0.6445  &gt;&gt; AP3 = polyfit(X, Y, 3)  AP3 =     -0.0726    0.1046    0.3196    0.3801  &gt;&gt; plot(X, Y, XS, polyval(AP2, XS), XS, polyval(AP3, XS))</pre> 	<pre data-bbox="787 447 1380 1123">public Point Task3(     Form form, Point location,     double[] x, double[] y,     double delta ) {     double[] xs = Range.Make(         x.First(), x.Last(), delta     );      double[] poly2 = Poly.Fit(x, y, 2);     double[] ys2 = Poly.Val(poly2, xs);      double[] poly3 = Poly.Fit(x, y, 3);     double[] ys3 = Poly.Val(poly3, xs);      Plot plot = new Plot();      plot.PlotSignalXY(x, y);     plot.PlotSignalXY(xs, ys2);     plot.PlotSignalXY(xs, ys3);      return plot.AddToForm(form, location); }</pre> 

**Висновки:** в ході роботи було здобуто навички використання математичного середовища MATLAB для апроксимації та інтерполяції функцій. Досліджену поведінку математичного середовища було відтворено за допомогою мови програмування C#.