4 ІНТЕРПОЛЯЦІЯ ФУНКЦІЙ У МАТЕМАТИЧНИХ МОДЕЛЯХ СКЛАДНИХ ОБ'ЄКТІВ І СИСТЕМ

1 Постановка задачі інтерполяції

Необхідно інтерполювати функцію $f(x) = 0.5x^2 + cos(2x)$ на інтервалі $[0.6; 1.1]$, користуючись вузловими точками, які обчислюються як $x_i = a + i * h$, де $h = (1.1 - 0.6)/10 = 0.05$, а $y_i = f(x)$. Таким чином було отримано наступні точки:

x_i	y_i
0.60	0.5424
0.65	0.4787
0.70	0.4150
0.75	0.3520
0.80	0.2908
0.85	0.2324
0.90	0.1778
0.95	0.1280
1.00	0.0839
1.05	0.0464

2 Правила складання поліномів Лагранжа і Ньютона

Поліном Лагранжа задається наступним чином: $L(x) = \sum_{j=0}^{n} y_j l_j(x)$, де

$$l_j(x) = \prod_{i=0, j\neq i}^{n} \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_n}{x_j - x_n} \qquad (1)$$

В нашому випадку $n = 9$. Поліном Ньютона задається наступним чином. Пряма(перша) інтерполяційн формула Ньютона має наступний вигляд:

$$P_n(x) = y_0 + q\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \ldots + \frac{q(q-1)\ldots(q-n+1)}{n!}\Delta^n y_0, \quad (2)$$

де $q = \frac{x - x_0}{h}$, $y_i = f_i$, а $\Delta^k y_0$ - скінчення різниці.

Зворотня(друга) інтерполяційн формула Ньютона має наступний вигляд:

$$P_n(x) = y_n + q\Delta y_{n-1} + \frac{q(q+1)}{2!}\Delta^2 y_{n-2} + \ldots + \frac{q(q+1)\ldots(q+n-1)}{n!}\Delta^n y_0 \qquad (3)$$

3 Результати роботи програми

Спершу програма генерує дані і зберігає їх у файл data.txt. Після чого на основі масиву даних проводиться інтерполяція многочленом Лагранжа та Ньютона. Програмно виводиться вигляд поліномів. Поліном Лагранджа має вигляд:

$$(-765231.400)(x-0.650)(x-0.700)(x-0.750)(x-0.800)(x-0.850)(x-0.900)$$
$$(x-0.950)(x-1.000)(x-1.050)+6079350.20475662(x-0.600)(x-0.700)(x-0.750)$$
$$(x-0.800)(x-0.850)(x-0.900)(x-0.950)(x-1.000)(x-1.050)+(-21077696.147)$$
$$(x-0.600)(x-0.650)(x-0.750)(x-0.800)(x-0.850)(x-0.900)(x-0.950)(x-1.000)$$
$$(x-1.050)+41717001.67913485(x-0.600)(x-0.650)(x-0.700)(x-0.800)(x-0.850)$$
$$(x-0.900)(x-0.950)(x-1.000)(x-1.050)+(-51697862.702)(x-0.600)(x-0.650)$$
$$(x-0.700)(x-0.750)(x-0.850)(x-0.900)(x-0.950)(x-1.000)(x-1.050)$$
$$+41316534.34746197(x-0.600)(x-0.650)(x-0.700)(x-0.750)(x-0.800)(x-0.900)$$
$$(x-0.950)(x-1.000)(x-1.050)+(-21072344.333)(x-0.600)(x-0.650)(x-0.700)$$
$$(x-0.750)(x-0.800)(x-0.850)(x-0.950)(x-1.000)(x-1.050)+$$
$$6499577.556139463(x-0.600)(x-0.650)(x-0.700)(x-0.750)(x-0.800)(x-0.850)$$
$$(x-0.900)(x-1.000)(x-1.050)+(-1064802.076)(x-0.600)(x-0.650)(x-0.700)$$
$$(x-0.750)(x-0.800)(x-0.850)(x-0.900)(x-0.950)(x-1.050)+$$
$$65472.86828944218(x-0.600)(x-0.650)(x-0.700)(x-0.750)(x-0.800)(x-0.850)$$
$$(x-0.900)(x-0.950)(x-1.000)$$

Для перевірки коректності роботи програми було обчислено значення поліномів у вузлах інтерполяції та точках посередині між вузлами і здійснено порівняння цих значень.

x	y інтерпольований	y справжній	помилка
0.60	0.5423577544766736	0.5423577544766736	0
0.62	0.5106348623953088	0.5106348623952687	4.007905118896815E-14
0.65	0.47874882862458734	0.47874882862458734	0
0.68	0.44681918709303303	0.4468191870930415	-8.493206138382448E-15
0.70	0.41496714290024084	0.41496714290024084	0
0.73	0.3833152693673698	0.3833152693673665	3.3306690738754696E-15
0.75	0.3519872016677028	0.3519872016677028	0
0.78	0.32110732780309037	0.3211073278030923	-1.942890293094024E-15
0.80	0.2908004776987111	0.2908004776987111	0
0.83	0.2611916111932677	0.26119161119326584	1.887379141862766E-15
0.85	0.2324055057044751	0.2324055057044751	0
0.88	0.20456644435050508	0.2045664443505077	-2.609024107869118E-15
0.90	0.17779790530691267	0.17779790530691267	0
0.93	0.1522222531754921	0.15222225317548688	5.218048215738236E-15
0.95	0.1279604331364963	0.1279604331364963	0
0.98	0.10513166864869543	0.10513166864871282	-1.7388868123191514E-14
1.00	0.08385316345285743	0.08385316345285743	0
1.03	0.06423980862339314	0.06423980862328688	1.062622212444353E-13
1.05	0.046403895400142336	0.046403895400142336	0

Також було оючислено середню похибку, яка рівна: $9.8532E-15$
Аналогічно для полінома Ньютона:

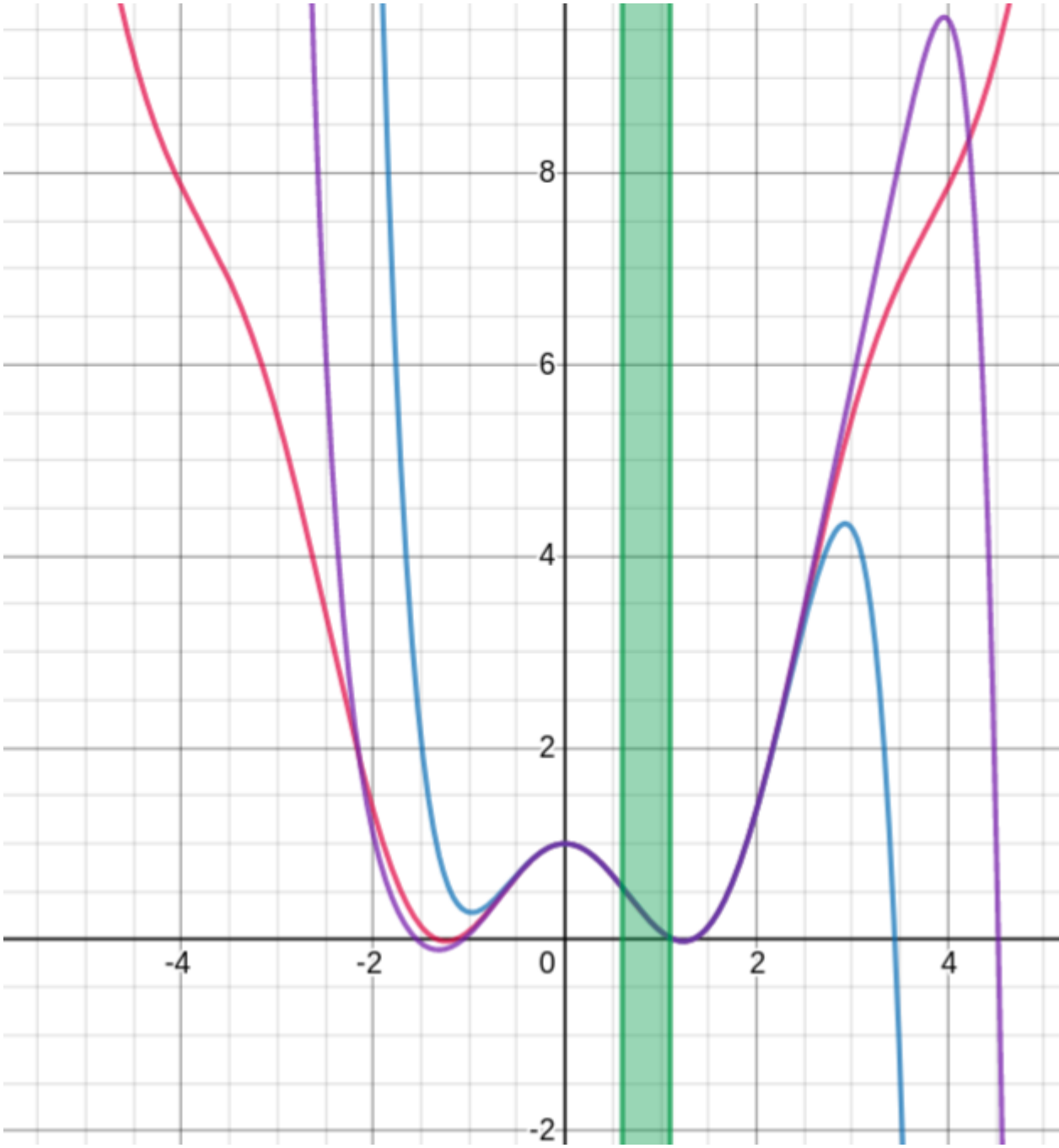
$$0.5423577544766736 + (-1.272)(x - 0.600) + (-0.035)(x - 0.600)(x - 0.650) +$$
$$1.2993391520916795(x - 0.600)(x - 0.650)(x - 0.700) + 0.11312271779380799(x - 0.600)$$
$$(x - 0.650)(x - 0.700)(x - 0.750) + (-0.264)(x - 0.600)(x - 0.650)(x - 0.700)(x - 0.750)$$
$$(x - 0.800) + (-0.006)(x - 0.600)(x - 0.650)(x - 0.700)(x - 0.750)(x - 0.800)$$
$$(x - 0.850) + 0.025317377875347595(x - 0.600)(x - 0.650)(x - 0.700)(x - 0.750)$$
$$(x - 0.800)(x - 0.850)(x - 0.900) + (-0.000)(x - 0.600)(x - 0.650)(x - 0.700)(x - 0.750)$$
$$(x - 0.800)(x - 0.850)(x - 0.900)(x - 0.950) + (-0.001)(x - 0.600)(x - 0.650)(x - 0.700)$$
$$(x - 0.750)(x - 0.800)(x - 0.850)(x - 0.900)(x - 0.950)(x - 1.000)$$

x	y інтерпольований	y справжній	помилка
0.60	0.5423577544766736	0.5423577544766736	0
0.62	0.5106348623953084	0.5106348623952687	3.9745984281580604E-14
0.65	0.47874882862458734	0.47874882862458734	0
0.68	0.446819187093033	0.4468191870930415	-8.548717289613705E-15
0.70	0.41496714290024084	0.41496714290024084	0
0.73	0.3833152693673698	0.3833152693673665	3.3306690738754696E-15
0.75	0.3519872016677028	0.3519872016677028	0
0.78	0.3211073278030903	0.3211073278030923	-1.9984014443252818E-15
0.80	0.2908004776987111	0.2908004776987111	0
0.83	0.2611916111932678	0.26119161119326584	1.942890293094024E-15
0.85	0.2324055057044751	0.2324055057044751	0
0.88	0.20456644435050506	0.2045664443505077	-2.6367796834847468E-15
0.90	0.17779790530691267	0.17779790530691267	0
0.93	0.15222225317549212	0.15222225317548688	5.245803791353865E-15
0.95	0.1279604331364963	0.1279604331364963	0
0.98	0.10513166864869553	0.10513166864871282	-1.7291723608536813E-14
1.00	0.08385316345285743	0.08385316345285743	0
1.03	0.06423980862339321	0.06423980862328688	1.0633161018347437E-13
1.05	0.046403895400142336	0.046403895400142336	0

Середня похибка: $9.8459E-15$

4 Результати порівняльного аналізу і висновки з роботи

По результатам роботи програми можемо бачити, що для заданої функції обидва методи інтерполяції є доволі точними. Інтерполяція поліномом Ньютона є трішки точніша. Також для наглядності побудуємо поліноми та початкову функцію на одному графіку. Отримаємо наступне:



На даному графіку червона лінія - це початкова функція, блакитна - поліном Лагранжа, фіолетова - поліном Ньютона. Зелена область -

інтервал $[0.6; 1.1]$, в якому проводилась інтерполяція.

5 Лістинг програми

```
using System;

namespace Interpolation
{
    public class Function
    {
        public static double a = 0.6;
        public static double b = 1.1;
        public static double step = (b-a)/10;
        public static double F(double x)
        {
            return 0.5 * x * x + Math.Cos(2*x);
        }
    }
}
```

```
using System;
using System.IO;
using System.Collections.Generic;

namespace Interpolation
{
    public class GenerateData
    {
        public static void Generate(Interpolation.Func f,
    double a, double b, double step, out double[] x, out double
    [] y)
        {
            double x0 = a;
            List<double> xi = new List<double>(); List<double>
    yi = new List<double>();
            while(x0<b)
            {
                xi.Add(x0);
                yi.Add(f(x0));
                x0+=step;
            }
            x = xi.ToArray(); y = yi.ToArray();
        }
        public static void SaveToFile(double[] x, double[] y,
    string file)
        {
            StreamWriter outp = new StreamWriter(file);
```

```
            outp.WriteLine("x_i\ty_i");
            for(int i = 0; i < x.Length; i++)
            {
                outp.WriteLine("{0:f2}\t{1:f4}", x[i], y[i]);
            }
            outp.Close();
        }
    }
}
```

```
using System;
using System.IO;
namespace Interpolation
{
    public class Interpolation
    {
        public delegate double Func(double x);
        public static double  EPS = 0.0001;
        public double[] Coef = null;
        public double[] x0, y0;

        public Interpolation(double[] x, double[] y){x0 = x; y0 = y; InitPolynomialCoeficients();}

        protected virtual void InitPolynomialCoeficients(){}
        public virtual double Interpolate(double x){return 0;}

        protected void CheckInterpolation(Interpolation.Func f, StreamWriter outp)
        {
            double x, y, y_interpolated; double avr_error = 0;
            outp.WriteLine("x&y interpolated&y real&error \\\\");
            for(int i = 0; i < x0.Length; i++)
            {
                y_interpolated = this.Interpolate(x0[i]);
                outp.WriteLine("{0:f2}&{1}&{2}&{3} \\\\", x0[i], y_interpolated, y0[i], y_interpolated - y0[i]);
                avr_error += Math.Abs(y_interpolated - y0[i]);
                if(i < x0.Length - 1)
                {
                    x = (x0[i] + x0[i + 1])/2; y = f(x);
                    y_interpolated = this.Interpolate(x);
                    outp.WriteLine("{0:f2}&{1}&{2}&{3} \\\\", x, y_interpolated, y, y_interpolated - y);
                    avr_error += Math.Abs(y_interpolated - y);
                }
            }
            outp.WriteLine("Average error: {0}", avr_error / (2
```

```
     * x0.Length - 1));
        }

    }
}
```

```
using System;
using System.IO;
namespace Interpolation
{
    public class InterpolationLagrange : Interpolation
    {
        public InterpolationLagrange(double[] x, double[] y): base(x, y){}

        protected override void InitPolynomialCoeficients()
        {
            int N = x0.Length;
            double[] coef = new double[N];
            for (int i = 0; i < N; i++)
            {
                coef[i] = y0[i];
                for (int j = 0; j < N; j++)
                    if (i != j)
                        coef[i] = coef[i] / (x0[i] - x0[j]);
            }
            Coef = coef;
        }
        public override string ToString()
        {
            int N = x0.Length;
            string sum = "";
            for (int i = 0; i < N; i++)
            {
                string add = "";
                for (int j = 0; j < N; j++)
                {
                    if (i != j)
                    {
                        if(x0[j] == 0) add += String.Format("x");
                        if(x0[j] > 0) add += String.Format("(x - {0:f3})", x0[j]);
                        if(x0[j] < 0) add += String.Format("(x - ({0:f3}))", x0[j]);
                        //if((j != (N - 1) && j != 0) && (!(j == (N - 2) && i == (N - 1)))) add += "*";
                    }
                }
```

```
39                  string c = Coef[i] >= 0 ? Coef[i].ToString() :
    String.Format("({0:f3})", Coef[i]);
40                  sum += String.Format("{0}{1}", c, add);
41                  if(i != (N - 1)) sum += "+";
42              }
43              return sum;
44          }
45          public override double Interpolate(double x)
46          {
47              // Calculate coeficients if they are not calculated
     yet
48              if(Coef == null)
49                  InitPolynomialCoeficients();
50              int N = x0.Length;
51              double s = 0, temp; int j, flag = 0;
52              for (int i = 0; i < N; i++)
53                  if (Math.Abs(x - x0[i]) < Interpolation.EPS)
54                  {
55                      s = y0[i];
56                      flag = 1;
57                  }
58              if (flag == 0)
59              {
60                  for (int i = 0; i < N; i++)
61                  {
62                      temp = Coef[i];
63                      for (j = 0; j < N; j++)
64                          if (i != j)
65                              temp *= x - x0[j];
66                      s += temp;
67                  }
68              }
69              return s;
70          }
71
72          // Method for testing
73          public static void TestInterpolation(double[] x0,
    double[] y0, Interpolation.Func f)
74          {
75              StreamWriter outp = new StreamWriter("
    output_lagrange.txt");
76              // Calculate and output Lagrange's poynomial
77              InterpolationLagrange lagr = new
    InterpolationLagrange(x0, y0);
78              outp.WriteLine(lagr.ToString());
79              outp.WriteLine();
80              // Check Lagrange's polynomial in (x0[i], y0[i])
    points and points between them
81              lagr.CheckInterpolation(f, outp);
```

```

            outp.Close();
        }

    }
}
```

```
using System;
using System.IO;
namespace Interpolation
{
    public class InterpolationNewton : Interpolation
    {
        public InterpolationNewton(double[] x, double[] y):base
    (x, y){}
        protected override void InitPolynomialCoeficients()
        {
            int N = x0.Length;
            double[] coef = new double[N];
            double a, temp; int j;
            coef[0] = y0[0];
            for (int i = 1; i < N; i++)
            {
                a = 1;
                temp = y0[i];
                for (j = 0; j < i; j++)
                {
                    temp -= a * coef[j];
                    a *= x0[i] - x0[j];
                }
                coef[i] = temp / a;
            }
            Coef = coef;
        }
        public override string ToString()
        {
            int N = x0.Length;
            string sum = "";
            string add = "";
            for (int i = 0; i < N; i++)
            {
                string c = Coef[i] >= 0 ? Coef[i].ToString() :
    String.Format("({0:f3})", Coef[i]);
                sum += String.Format("{0}{1}", c, add);
                if(i != (N - 1)) sum += "+";

                if(x0[i] == 0) add += String.Format("x");
                if(x0[i] > 0) add += String.Format("(x - {0:f3
    })", x0[i]);
```

```
40                  if(x0[i] < 0) add += String.Format("(x - ({0:f3
    }))", x0[i]);
41                  //if((j != (N - 1) && j != 0) && (!(j == (N -
    2) && i == (N - 1)))) add += "*";
42
43              }
44              return sum;
45          }
46          public override double Interpolate(double x)
47          {
48              // Calculate coeficients if they are not calculated
     yet
49              if(Coef == null)
50                  InitPolynomialCoeficients();
51              int N = x0.Length;
52
53              double s = Coef[0];
54              double a = 1;
55              int flag = 0;
56              for (int i = 0; i < N; i++)
57                  if (Math.Abs(x - x0[i]) < Interpolation.EPS)
58                  {
59                      s = y0[i];
60                      flag = 1;
61                  }
62              if (flag == 0)
63              {
64                  for (int i = 1; i < N; i++)
65                  {
66                      a *= x - x0[i - 1];
67                      s += Coef[i] * a;
68                  }
69              }
70              return s;
71          }
72
73          // Method for testing
74          public static void TestInterpolation(double[] x0,
    double[] y0, Interpolation.Func f)
75          {
76              StreamWriter outp = new StreamWriter("output_newton
    .txt");
77              // Calculate and output Newtons's poynomial
78              InterpolationNewton newt = new InterpolationNewton(
    x0, y0);
79              outp.WriteLine(newt.ToString());
80              outp.WriteLine();
81              // Check Newton's polynomial in (x0[i], y0[i])
    points and points between them
```

```
            double x, y, y_interpolated;

            newt.CheckInterpolation(f, outp);

            outp.WriteLine("Interpolation for x4 + 0.021:");
            outp.WriteLine("x;y_interpolated;y_real;difference
    ");
            x = x0[4] + 0.021; y = f(x);
            y_interpolated = newt.Interpolate(x);
            outp.WriteLine("{0}\t{1}\t{2}\t{3}", x,
    y_interpolated, y, y_interpolated - y);
            outp.WriteLine("Interpolation for x7 - 0,0146:");
            outp.WriteLine("x;y_interpolated;y_real;difference
    ");
            x = x0[7] - 0.0146; y = f(x);
            y_interpolated = newt.Interpolate(x);
            outp.WriteLine("{0}\t{1}\t{2}\t{3}", x,
    y_interpolated, y, y_interpolated - y);


            outp.Close();
        }

    }
}
```

```
  using   System;

namespace Interpolation
{
    class Program
    {
        static void Main(string[] args)
        {
            // Generate x_i and y_i and save them to file data.
    txt
            double[] x, y;
            GenerateData.Generate(Function.F, Function.a,
    Function.b, Function.step, out x, out y);
            GenerateData.SaveToFile(x, y, "data.txt");

            InterpolationLagrange.TestInterpolation(x, y,
    Function.F);
            InterpolationNewton.TestInterpolation(x, y,
    Function.F);
        }
    }
}
```