**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра вычислительной математики**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**”Аппроксимация непрерывной функции с помощью кубического сплайна”**

Возовикова Никиты Александровича

студента 2 курса группы 10

специальности «Компьютерная Безопасность»

дневной формы получения

высшего образования

Научный руководитель:

Доцент

Никифоров Иван Васильевич

Минск, 2021

**ОГЛАВЛЕНИЕ**

**1. Постановка задачи**

**2. Краткие теоретические сведения**

**3. Листинг программы**

**4. Результаты**

**5. Выводы**

**Постановка задачи**

Функция на отрезке задана таблицей своих значений с шагом . Определить наибольшую погрешность аппроксимации, если:

Построить графики функций.

**Краткие теоретические сведения**

Интерполяция кубическими сплайнами является частным случаем кусочно-полиномиальной интерполцией. В этом специальном случае между любыми двумя соседними узлами функция интерполируется кубическим полиномом. его коэффициенты на каждом интервале определяются из условий сопряжения в узлах:

f_i=y_i,

f'(x_i-0)=f'(x_i+0),

f''(x_i-0)=f''(x_i+0), i=1, 2, \cdots, n-1.

Кроме того, на границе при x=x_0 и x=x_n ставятся условия

f''(x_0)=0, f''(x_n)=0. (2)

Будем искать кубический полином в виде

f(x)=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3, x_{i-1}\le \x\le \x_i. (3)

Из условия f_i=y_i имеем

f(x_{i-1})=a_i=y_{i-1},

f(x_i)=a_i+b_ih_i+c_ih_i^2+d_ih_i^3=y_i,

h_i=x_i-x_{i-1}, i=1, 2, \cdots, n-1.(4)

Вычислим производные:

f'(x)=b_i+2c_i(x-x_{i-1})+3d_i(x-x_{i-1})^2,

f''(x)=2c_i+6d_i(x-x_{i-1}), x_{i-1}\le \x\le \x_i,

и потребуем их непрерывности при x=x_i:

b_{i+1}=b_i+2c_ih_i + 3d_ih_i^2,

c_{i+1}=c_i+3d_ih_i, i=1, 2, \cdots, n-1. (5)

Общее число неизвестных коэффициентов, очевидно, равно 4n, число уравнений [(4)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:4" \o ") и [(5)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8#eq:5) равно 4n-2. Недостающие два уравнения получаем из условия [(2)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:2" \o ") при x=x_0 и x=x_n:

c_1=0, c_n+3d_nh_n=0.

Выражение из [(5)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:5" \o ") d_i=\frac{c_{i+1}-c_i}{3h_i}, подставляя это выражение в [(4)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:4" \o ") и исключая a_i=y_{i-1}, получим

b_i=\[\frac{y_i-y_{i-1}}h_i\]-\frac{1}{3}h_i(c_{i+1}+2c_i),  i=1, 2, \cdots, n-1,

b_n=\[\frac{y_n-y_{n-1}}h_n\]-\frac{2}{3}h_nc_n,.

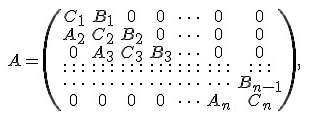
Подставив теперь выражения для b_i, b_{i+1} и d_i в первую формулу [(5)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:5" \o "), после несложных преобразований получаем для определения c_i разностное уравнение второго порядка

h_ic_i+2(h_i+h_{i+1})c_{i+1}+h_{i+1}c_{i+2}=3\left(\frac{y_{i+1}-y_i}{h_{i+1}} - \frac{y_i-y_{i-1}}{h_i}\right), i=1, 2, \cdots, n-1.(6)

С краевыми условиями

c_1=0, c_{n+1}=0. (7)

Условие c_{n+1}=0 эквивалентно условию c_n+3d_nh_n=0 и уравнению c_{i+1} = c_i+d_ih_i. Разностное уравнение [(6)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:6" \o ") с условиями [(7)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:7" \o ") можно решить методом прогонки, представив в виде системы линейных алгебраических уравнений вида ~A*x=F, где вектор x соответствует вектору \{c_i\}, вектор F поэлементно равен правой части уравнения [(6)](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8" \l "eq:6" \o "), а матрица ~A имеет следующий вид:



где

A_i=h_i,  i=2, \cdots, n,  B_i = h_{i+1},  i=1, \cdots, n-1 и C_i=2(h_i+h_{i+1}), i =1, \cdots, n.

**Листинг программы**

**#include "stdafx.h"**

**using namespace System;**

**#include "stdafx.h"**

**#include "stdio.h"**

**#include "string.h"**

**#include "math.h"**

**class knot{**

**public:**

**double x,f;**

**void Add(double arg,double func)**

**{**

**x=arg;**

**f=func;**

**}**

**knot(){}**

**};**

**class vector{**

**public:**

**double\* x;**

**void Add(int m)**

**{**

**x = new double[m];**

**}**

**vector()**

**{**

**}**

**};**

**knot\* KnotArray;**

**int n;**

**double\*\* Coef;**

**double\* b;**

**void SolveTriDiag(double\*\* TDM, double\* F)**

**{**

**double\* alph = new double[n-1];**

**double\* beta = new double[n-1];**

**int i;**

**alph[0] = - TDM[2][0]/TDM[1][0];**

**beta[0] = F[0]/TDM[1][0];**

**for (i=1; i<n-1; i++)**

**{**

**alph[i] = -TDM[2][i]/(TDM[1][i] + TDM[0][i]\*alph[i-1]);**

**beta[i] = (F[i]-TDM[0][i]\*beta[i-1])/(TDM[1][i] + TDM[0][i]\*alph[i-1]);**

**}**

**b[n-1] = (F[n-1]-TDM[0][n-1]\*beta[n-2])/(TDM[1][n-1] + TDM[0][n-1]\*alph[n-2]);**

**for (i=n-2; i>-1; i--)**

**{**

**b[i] = b[i+1] \* alph[i] + beta[i];**

**}**

**}**

**int BuildSpline()**

**{**

**double\* a = new double[n-1];**

**double\* c = new double[n-1];**

**double\* d = new double[n-1];**

**double\* delta = new double[n-1];**

**double\* h = new double[n-1];**

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

**b = new double[n];**

**TriDiagMatrix[0] = new double[n];**

**TriDiagMatrix[1] = new double[n];**

**TriDiagMatrix[2] = new double[n];**

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

**double x3,xn;**

**int i;**

**if (n<3)**

**return -1;**

**x3 = KnotArray[2].x - KnotArray[0].x;**

**xn = KnotArray[n-1].x - KnotArray[n-3].x;**

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

**{**

**a[i] = KnotArray[i].f;**

**h[i] = KnotArray[i+1].x - KnotArray[i].x;**

**delta[i] = (KnotArray[i+1].f - KnotArray[i].f)/h[i];**

**TriDiagMatrix[0][i] = i>0?h[i]:x3;**

**f[i] = i>0?3\*(h[i]\*delta[i-1] + h[i-1]\*delta[i]):0;**

**}**

**TriDiagMatrix[1][0] = h[0];**

**TriDiagMatrix[2][0] = h[0];**

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

**{**

**TriDiagMatrix[1][i] = 2\*(h[i] + h[i-1]);**

**TriDiagMatrix[2][i] = h[i];**

**}**

**TriDiagMatrix[1][n-1] = h[n-2];**

**TriDiagMatrix[2][n-1] = xn;**

**TriDiagMatrix[0][n-1] = h[n-2];**

**i = n-1;**

**f[0] = ((h[0]+2\*x3)\*h[1]\*delta[0] + powf(h[0],2)\*delta[1])/x3;**

**f[n-1]=(powf(h[i-1],2)\*delta[i-2]+(2\*xn+h[i-1])\*h[i-2]\*delta[i-1])/xn;**

**SolveTriDiag(TriDiagMatrix,f);**

**Coef = new double\*[4];**

**Coef[0] = new double[n-1];**

**Coef[1] = new double[n-1];**

**Coef[2] = new double[n-1];**

**Coef[3] = new double[n-1];**

**int j;**

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

**{**

**d[j] = (b[j+1]+b[j]-2\*delta[j])/(h[j]\*h[j]);**

**c[j] = 2\*(delta[j]-b[j])/h[j]-(b[j+1]-delta[j])/h[j];**

**Coef[j][0] = a[j];**

**Coef[j][1] = b[j];**

**Coef[j][2] = c[j];**

**Coef[j][3] = d[j];**

**}**

**}**

**double Interpolate(double x)**

**{**

**//double result;**

**int i=0;**

**while (KnotArray[i].x < x)**

**i++;**

**i--;**

**return Coef[i][0] + Coef[i][1]\*(x-KnotArray[i].x) + Coef[i][2]\*powf((x-KnotArray[i].x),2) + Coef[i][3]\*powf((x-KnotArray[i].x),3);**

**}**

**int Load\_Data()**

**{**

**printf("Input filename with data\n");**

**char FileName[20];**

**int i=0;**

**double ii;**

**FILE \*File;**

**scanf("%s",&FileName);**

**if (!fopen\_s(&File,FileName,"r"))**

**{**

**printf("file is opened\n");**

**}**

**else**

**{**

**printf("%s: file doesn't exist\n",FileName);**

**return -1;**

**}**

**double x,f;**

**fscanf(File,"%d",&n);**

**KnotArray = new knot[n];**

**while (!feof(File))**

**{**

**fscanf(File,"%lf%lf",&x,&f);**

**KnotArray[i].Add(x,f);**

**i++;**

**if (i==n+1)**

**return -1;**

**}**

**fclose(File);**

**return 1;**

**}**

**int main(array<System::String ^> ^args)**

**{**

**double x=0;**

**if(Load\_Data()!=-1)**

**{**

**BuildSpline();**

**scanf("%lf",&x);**

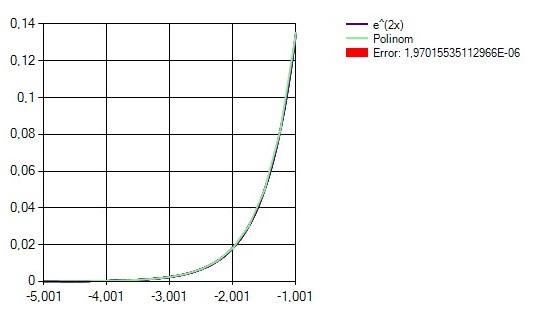
**printf("x: %lf\nans: %lf\n",x,Interpolate(x));**

**scanf("%lf",&x);**

**}**

**return 0;}**

**Результаты**

****

*Рис 1.1 График функции и приближения функции*

Наибольшая погрешность аппроксимации: **1,47045233311926E-06**

**Выводы**

По окончанию выполнения лабораторной работы было реазизованно приближение функции с помощью кубического сплайна и выведены соотвествующие графики. В результате решения была найдена наибольшая погрешность аппроксимации равная 1,47045233311926E-06

Сплайны имеют многочисленные применения как в математической теории, так и в прикладной математике (в частности, в разнообразных вычислительных программах).

В частности, сплайны двух переменных интенсивно используются для задания поверхностей в различных системах компьютерного моделирования.