МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский физико-технический институт
(национальный исследовательский университет)

Интерполяция. Сплайны

Лабораторная работа №1 по курсу
Вычислительная математика
Вариант 5

Выполнил: студент группы Б04-856.
Крылов А. А.

г. Долгопрудный
2020 год

Содержание

1. Цели и задачи работы

Цель работы: построить аналитические приближения функций по таблице значений различными методами

Задачи:

- Произвести алгебраическою интерполяцию;
- Произвести кусочно-многочленную интерполяцию(построить сплайн)

2. Используемые методы

2.1. Алгебраическая интерполяция в форме Ньютона

Задача интерполяции ставится следующим образом: для заданной системы $\{x_k, f_k\}_{k=0}^{N}$ предложить обобщенный многочлен, который в узлах интерполяции принимает значения функции. При этом число нулей многочлена должно быть не больше, чем N. Одна из систем функций, позволяющей решить задачу и удовлетворить условию выше является система $\{1, x, x^2, ..., x^N, ...\}$

Известны различные формы записи интерполяционного многочлена. В данной работе применяется запись в форме Ньютона:

$$P_N(x) = \sum b_n \prod (x - x_i) \tag{1}$$

Здесь коэффициенты b_n определяются как:

$$b_0 = f_0; b_1 = f(x_0, x_1) = \frac{f_1 - f_0}{x_1 - x_0}; ...b_N = \frac{f(x_1, ..., x_N) - f(x_0, x_1, ..., x_{N-1})}{x_N - x_0} = f(x_0, x_1, ..., x_N) \tag{2}$$

Пожалуй, это наиболее простой метод для реализации на ЭВМ, поэтому она применена при реализации задачи

2.2. Сплайны

Сплайн – функция, которая вместе с несколькими производными непрерывна на всем заданном отрезке $[a, b]$, а на каждом частичном отрезке $[x_i, x_{i+1}]$ в отдельности является некоторым многочленом. Такая функция позволяет более точно приблизить результат при меньших затратах вычислительной мощности.

Задача построения сплайна

$$S_{3i}(x) = (a_0)_i + (a_1)_i x + (a_2)_i x^2 + (a_3)_i x^3, \tag{3}$$

который будет интерполировать многочлен P_N сводится к решению систем уравнений на различных отрезках $[x_i, x_{i+1}]$:

$$\begin{gathered}(a_0)_i + (a_1)_i x_i + (a_2)_i x_i^2 + (a_3)_i x_i^3 = P_N(x_i), \\ (a_0)_i + (a_1)_i x_{i+1} + (a_2)_i x_{i+1}^2 + (a_3)_i x_{i+1}^3 = P_N(x_{i+1}) \\ (a_1)_i + 2(a_2)_i x_i + 3(a_3)_i x_i^2 = P'_N(x_i), \\ (a_1)_i + 2(a_2)_i x_{i+1} + 3(a_3)_i x_{i+1}^2 = P'_N(x_{i+1})\end{gathered} \tag{4}$$

Производную P_N также просто найти, отсюда можно получить коэффициенты сплайна на $[x_i, x_{i+1}]$

3. Результаты

Для решения поставленной задачи была написана программа на языке C++, код которой приведен в приложении. В ней сначала происходит расчет коэффицентов b_n для многочлена Ньютона, затем его вид упроращается. Далее расчитываются коэффициенты сплайна на отрезках, концами которых служат узлы сетки, заданной таблично. В конце производится расчет значения интерполяционного многочлена Ньютона и сплайна в заданной точке. Результат работы программы для таблицы значений варианта 5 со значениями интерполяционных функций в точке $x = 0,87$ также приведен в приложении.

4. Вывод

Разработана программа для построения интерполяционного многочлена и функции сплайна для табличной функции. Изучены методы интерполяции.

5. Приложения

5.1. Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#define FTYPE long double

FTYPE sourceData[7][2] = {
{0.5236, 0.0010},
{0.87267, 0.00112},
{1.22173, 0.00854},
{1.57080, 0.03018},
{1.91986, 0.10659},
{2.26893, 0.3207},
{2.61799, 0.85128}
};
int length = 7;
//get b coefficents for Newton polynom
FTYPE getB(int argc, int* argv, FTYPE source[7][2]) {
if (argc == 1) {
return source[argv[0]][1];
}
else {
int xs1[7];
int xs2[7];
int lensub = argc - 1;
for (int i = 0; i < lensub; i++) {
xs1[i] = argv[i];
}
for (int i = 1; i < argc; i++) {
xs2[i - 1] = argv[i];
}
FTYPE fa = getB(lensub, xs2, source);
FTYPE fb = getB(lensub, xs1, source);

return (fa - fb) / (source[argc - 1][0] - source[0][0]);
}
}
//get value in point for spline function
FTYPE evaluate(FTYPE* function, FTYPE x) {
FTYPE result = 0;
for (int i = 0; i < length; i++) {
result += function[i] * powl(x, (FTYPE)i);
}
return result;
}

FTYPE* D(FTYPE* function) {
```

```
FTYPE* result = (FTYPE*)malloc(sizeof(FTYPE) * 5);
for (int i = 1; i < 7; i++) {
result[i - 1] = function[i] * (i);
}
return result;
}

FTYPE* calculateSplineCoefs(FTYPE* function, FTYPE points[7][2], int i) {
FTYPE xi = points[i][0];
FTYPE xi1 = points[i + 1][0];

FTYPE* pnd = D(function);

FTYPE fi = evaluate(function, xi);
FTYPE fi1 = evaluate(function, xi1);
FTYPE pdi = evaluate(pnd, xi);
FTYPE pdi1 = evaluate(pnd, xi1);
FTYPE dfi = fi1 - fi;
FTYPE H = xi1 - xi;
FTYPE dh3 = powl(xi1 - xi, 3.0);

FTYPE a3 = ((pdi1 * H - 2.0 * dfi + pdi * H) / dh3);
FTYPE a2 = ((-pdi1 * H * (xi1 + 2.0 * xi) +
3.0 * dfi * (xi1 + xi) - pdi * H * (xi + 2.0 * xi1)) / dh3);
FTYPE a1 = ((pdi1 * xi * (xi + 2.0 * xi1) * H -
6.0 * dfi * xi * xi1 + pdi * xi1 * (xi1 + 2.0 * xi) * H) / dh3);
FTYPE a0 = ((-pdi1 * xi * xi * xi1 * H + fi1 * xi * xi * (3.0 * xi1 - xi) +
fi * xi1 * xi1 * (xi1 - 3.0 * xi) - pdi * xi * xi1 * xi1 * H) / dh3);

FTYPE* res = (FTYPE *)malloc(4 * sizeof(FTYPE));
res[0] = a0;
res[1] = a1;
res[2] = a2;
res[3] = a3;
return res;
}

FTYPE** getAllSplineCoefs(FTYPE* function, FTYPE points[7][2]) {
FTYPE** coefs = (FTYPE**)malloc(sizeof(FTYPE*) * length);
for (int i = 0; i < length; i++) {
coefs[i] = (FTYPE *)malloc(sizeof(FTYPE) * 4);
}
for (int i = 0; i < length; i++) {
for (int j = 0; j < 4; j++) {
coefs[i][j] = 0.0;
}
}
FTYPE* line = NULL;
for (int i = 0; i < length - 1; i++) {
line = calculateSplineCoefs(function, points, i);
```

```
for (int j = 0; j < 4; j++) {
coefs[i][j] = line[j];
}
}
return coefs;
}

FTYPE calculateInPoint(FTYPE** splineCoefs, FTYPE points[7][2], FTYPE x) {
FTYPE result = 0;
bool inBounds = false;
for (int i = 0; i < length - 1; i++) {
if ((x >= sourceData[i][0]) && (x <= sourceData[i + 1][0])) {
inBounds = true;
for (int j = 0; j < 4; j++) {
result += splineCoefs[i][j] * powl(x, j);
}
}
}
if (inBounds) {
return result;
}
else {
printf("Error: Not in spline bounds\n");
return 0;
}
}

static void makeCache(long double(*cache)[2]) {
for (int i = 0; i < length / 2 + 1; i++) {
cache[i][0] = sourceData[length - 1 - i][0];
cache[length - 1 - i][0] = sourceData[i][0];
cache[i][1] = sourceData[length - 1 - i][1];
cache[length - 1 - i][1] = sourceData[i][1];
}
}

static void primaryInterpolation(long double(*cache)[2], long double* stringOfCoeffs) {
for (int i = 0; i < length; i++) {
int xs[7];
int len = i + 1;
for (int j = 0; j <= i; j++) {
xs[j] = j;
}
printf("%Lf", getB(len, xs, cache));
stringOfCoeffs[i] = getB(len, xs, cache);
for (int j = 0; j < i; j++) {
printf("(x-%Lf)", cache[j][0]);
}
if (i < length - 1) {
printf("+");
```

```
}
}
}

static void simplifyInterpolation(long double(*cache)[2],
long double* simpleFormOfPolynom,
long double* stringOfCoeffs) {
FTYPE brackets[7][2];
for (int i = 0; i < 6; i++) {
brackets[i][0] = (-1) * cache[i][0];
brackets[i][1] = 1.0;
}
FTYPE last[7];
for (int i = 0; i < 7; i++) {
last[i] = 0.0;
}
FTYPE lastW[7];
for (int i = 0; i < 7; i++) {
lastW[i] = last[i];
}
last[0] = 1;
for (int i = 0; i < length; i++) {
for (int j = 0; j < length; j++) {
simpleFormOfPolynom[j] += stringOfCoeffs[i] * last[j];
}
for (int br = i; br <= i; br++) {
for (int k = 0; k < 2; k++) {
for (int cf = 0; cf < length; cf++) {
lastW[cf + k] += last[cf] * brackets[br][k];
}
}
for (int j = 0; j < length; j++) {
last[j] = lastW[j];
lastW[j] = 0.0;
}
}
}
for (int i = 0; i < length; i++) {
if (i == 0) {
printf("%Lf", simpleFormOfPolynom[i]);
}
else {
printf("%Lf*x^%d", simpleFormOfPolynom[i], i);
}
if ((i < length - 1) && (simpleFormOfPolynom[i + 1] >= 0)) {
printf("+");
}
}
}
```

```
int main(int argc, const char* argv[]) {
printf("Computational Mathematics Lab #2\nVariant 5\n\n");
FTYPE cache[7][2];
FTYPE simpleFormOfPolynom[7];
for (int i = 0; i < 7; i++) {
simpleFormOfPolynom[i] = 0.0;
}
FTYPE stringOfCoeffs[7];
for (int i = 0; i < 7; i++) {
stringOfCoeffs[i] = 0.0;
}
makeCache(cache);
printf("Input data:\n");
for (int i = 0; i < length; i++) {
printf("{%Lf, %Lf}\n", cache[i][0], cache[i][1]);
}
printf("\nInterpolation:\nP_N(x)=");
primaryInterpolation(cache, stringOfCoeffs);
printf("\n\nSimplified:\n");
simplifyInterpolation(cache, simpleFormOfPolynom, stringOfCoeffs);
printf("\n");
printf("\nSpline coefs:\n");
for (int i = 0; i < length; i++) {
cache[i][0] = sourceData[i][0];
cache[i][1] = sourceData[i][1];
}
FTYPE** coefs = getAllSplineCoefs(simpleFormOfPolynom, cache);
for (int i = 0; i < length - 1; i++) {
printf("{");
for (int j = 0; j < 4; j++) {
if (j < 3) {
printf("%Lf,", coefs[i][j]);
}
else {
printf("%Lf", coefs[i][j]);
}
}
printf("}");
if (i < length - 2) {
printf(",\n");
}
}
printf("\nInput x point: ");
FTYPE x;
scanf_s("%Lf", &x);
FTYPE result = calculateInPoint(coefs, sourceData, x);
FTYPE intRes = evaluate(simpleFormOfPolynom, x);
printf("I(%Lf)=%Lf\nS(%Lf)=%Lf\n", x, intRes, x, result);
system("pause");
return 0;
```

```
}
```

5.2. Вывод программы

```
Computational Mathematics Lab #2
Variant 5

Input data:
{2.617990, 0.851280}
{2.268930, 0.320700}
{1.919860, 0.106590}
{1.570800, 0.030180}
{1.221730, 0.008540}
{0.872670, 0.001120}
{0.523600, 0.001000}

Interpolation:
P_N(x)=0.851280+
1.520025(x-2.617990)+
1.298662(x-2.617990)(x-2.268930)+
0.700540(x-2.617990)(x-2.268930)(x-1.919860)+
0.268979(x-2.617990)(x-2.268930)(x-1.919860)(x-1.570800)+
0.085966(x-2.617990)(x-2.268930)(x-1.919860)(x-1.570800)(x-1.221730)+
0.034328(x-2.617990)(x-2.268930)(x-1.919860)(x-1.570800)(x-1.221730)(x-0.872670)

Simplified:
0.189553-0.954546*x^1+1.863279*x^2-1.831249*x^3+0.975689*x^4-0.273513*x^5+0.034328*x^6

Spline coefs:
{0.109944,-0.430923,0.537615,-0.213971},
{-0.047494,0.124229,-0.114494,0.041209},
{-0.250309,0.626796,-0.529572,0.155473},
{-0.847065,1.753146,-1.238118,0.304025},
{-3.457170,5.791910,-3.321052,0.662071},
{-12.302514,17.412501,-8.409700,1.404811}
Input x point: 0.87
I(0.870000)=0.001061
S(0.870000)=0.001061
```