МГТУ им. Н. Э. Баумана

Отчет по лабораторной работе №3 по предмету «Вычислительные алгоритмы»
«Построение и программная реализация алгоритма сплайн-интерполяции табличных функций»

Выполнила Параскун София,
ИУ7-44Б
Проверил
Градов В. М.

Москва, 2021 г.

Цель работы: получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

1. Исходные данные

1. Таблица функции с количеством узлов N. Задать с помощью формулы $y = x^2$ в диапазоне [0..10] с шагом 1.

2. Значения аргумента x в первом интервале, например, при x = 0.5 и в середине таблицы, например, при x = 5.5. Сравнить с точным значением.

2. Код программы

Программа состоит из 5 файлов, которые собираются в app.exe. Задаваемая таблица функции хранится внутри программы.

main.c

```
#include "spline.h"
int main()
{
   int res = EXIT_SUCCESS;
   struct Data data;
   res = dataInput(&data);
   if (!res)
   {
      for (int i = 0; i < 2 && !res; i++)
      {
         res = xInput(&data);
         if (!res)
         {
            res = splineCalc(&data);
            if (!res)
               resOut(data);
         }
      }
   }
   return res;
}
```

struct.h

```
#ifndef STRUCT_H
#define STRUCT_H
```

```
#include <stdio.h>
#include <stdlib.h>

#define MEMERR -1
#define INPUTERR -2

struct Data
{
    int N;
    int *func;
    float x;
    int ind;
    float *h;
    float *aCoef;
    float *bCoef;
    float *cCoef;
    float *dCoef;
    float res;
};

int dataInput(struct Data *data);
int xInput(struct Data *data);
void resOut(struct Data data);

#endif // STRUCT_H
```

struct.c

```
#include "struct.h"
int dataInput(struct Data *data)
{
    int statusCode = EXIT_SUCCESS;
    data->N = 10;
    data->func = malloc(sizeof(int) * (data->N + 1));
    if (!data->func)
        statusCode = MEMERR;
    else
    {
        for (int i = 0; i < data->N + 1; i++)
                data->func[i] = i * i;
    }
    return statusCode;
}

int xInput(struct Data *data)
{
    int statusCode = EXIT_SUCCESS;
    printf("Input x: ");
```

```
    int rc = scanf("%f", &data->x);
    if (!rc)
        statusCode = INPUTERR;
    else
    {
        if ((data->x < 0) || (data->x > data->N))
            statusCode = INPUTERR;
        else
            for (int i = 0; i < data->N; i++)
                if (data->x > i && data->x < i + 1)
                    data->ind = i + 1;
    }
    return statusCode;
}

void resOut(struct Data data)
{
    printf("Spline-interpolation equal %.4f\n", data.res);
    printf("Exact value equal %.4f\n", data.x * data.x);
    printf("Innacuracy is %.2f%%\n", (data.res / (data.x * data.x) - 1) * 100);
}
```

spline.h

```
#ifndef SPLINE_H
#define SPLINE_H

#include "struct.h"

int splineCalc(struct Data *data);

#endif // SPLINE_H
```

spline.c

```
#include "spline.h"
int coefAlloc(struct Data *data);
int hCalc(struct Data *data);
void findACoef(struct Data *data);
void findBCoef(struct Data *data);
void findCCoef(struct Data *data);
void findDCoef(struct Data *data);
void funcCalc(struct Data *data);

int splineCalc(struct Data *data)
{
    int statusCode = EXIT_SUCCESS;
    statusCode = coefAlloc(data);
```

```
    if (!statusCode)
    {
        statusCode = hCalc(data);
        if (!statusCode)
        {
            findCCoef(data);
            findACoef(data);
            findBCoef(data);
            findDCoef(data);

            funcCalc(data);
        }
    }
    return statusCode;
}

int coefAlloc(struct Data *data)
{
    int statusCode = EXIT_SUCCESS;
    data->aCoef = malloc((data->N + 1) * sizeof(float));
    if (!data->aCoef)
        statusCode = MEMERR;
    else
    {
        data->bCoef = malloc((data->N + 1) * sizeof(float));
        if (!data->bCoef)
            statusCode = MEMERR;
        else
        {
            data->cCoef = malloc((data->N + 2) * sizeof(float));
            if (!data->cCoef)
                statusCode = MEMERR;
            else
            {
                data->dCoef = malloc((data->N + 1) * sizeof(float));
                if (!data->dCoef)
                    statusCode = MEMERR;
            }
        }
    }
    return statusCode;
}

int hCalc(struct Data *data)
{
    int statusCode = EXIT_SUCCESS;
    data->h = malloc((data->N + 1) * sizeof(float));
    if (!data->h)
        statusCode = MEMERR;
```

```
    else
    {
        for (int i = 0; i < data->N + 1; i++)
            data->h[i] = 1;
    }
    return statusCode;
}

void findCCoef(struct Data *data)
{
    float e[data->N + 2];
    float n[data->N + 2];
    float f[data->N + 1];

    for (int i = 2; i < data->N + 1; i++)
        f[i] = 3 * ((data->func[i] - data->func[i - 1]) / data->h[i] - (data->func[i - 1] - data->func[i - 2]) /
data->h[i - 1]);

    e[2] = 0, n[2] = 0;
    for (int i = 3; i < data->N + 2; i++)
    {
        e[i] = data->h[i - 1] / (data->h[i - 2] * e[i - 1] + 2 * (data->h[i - 2] + data->h[i - 1]));
        n[i] = (f[i - 1] - data->h[i - 2] * n[i - 1]) / (data->h[i - 2] * e[i - 1] + 2 * (data->h[i - 2] + data->h[i -
1]));
    }

    data->cCoef[1] = 0;
    data->cCoef[data->N + 1] = 0;
    for (int i = data->N; i > 1; i--)
        data->cCoef[i] = e[i + 1] * data->cCoef[i + 1] + n[i + 1];
}

void findACoef(struct Data *data)
{
    for (int i = 1; i < data->N + 1; i++)
        data->aCoef[i] = data->func[i - 1];
}

void findBCoef(struct Data *data)
{
    for (int i = 1; i < data->N; i++)
        data->bCoef[i] = (data->func[i] - data->func[i - 1]) / data->h[i] - data->h[i] * (data->cCoef[i + 1] -
2 * data->cCoef[i]) / 3;
    data->bCoef[data->N] = (data->func[data->N] - data->func[data->N - 1]) / data->h[data->N] - data-
>h[data->N] * 2 * data->cCoef[data->N] / 3;
}

void findDCoef(struct Data *data)
```

```
{
   for (int i = 1; i < data->N; i++)
      data->dCoef[i] = (data->cCoef[i + 1] - data->cCoef[i]) / 3 / data->h[i];
   data->dCoef[data->N] = -data->cCoef[data->N] / 3 / data->h[data->N];
}

void funcCalc(struct Data *data)
{
   float dif = data->x - data->ind + 1;
   data->res = data->aCoef[data->ind] + data->bCoef[data->ind] * dif + data->cCoef[data->ind] * dif *
dif + data->dCoef[data->ind] * dif * dif * dif;
}
```

3. Результаты работы

Результатом будут значения y(x), полученные сплайн-интерполяцией, в сравнении с полиномом Ньютона 3-й степени.

X	Сплайн-интерполяция	Полином Ньютона, 3 степень	Точное значение	Погрешность сплайна
0.5	0.2682	0.2500	0.2500	7.3%
1.5	3.3273	2.2500	2.2500	47.8%
2.5	7.0773	6.2500	6.2500	13.2%
3.5	13.1363	12.2500	12.2500	7.2%
4.5	21.1223	20.2500	20.2500	4.3%
5.5	31.1256	30.2500	30.2500	2.9%
6.5	43.1248	42.2500	42.2500	2.1%
7.5	57.1248	56.2500	56.2500	1.6%
8.5	73.1240	72.2500	72.2500	1.2%
9.5	90.3568	90.2500	90.2500	0.1%

Как видим в начале таблицы рядом с 0 погрешность является достаточно заметной, это связано с обратным ходом при определении коэффициентов C.

4. Вопросы при защите лабораторной работы

1. Получить выражение для коэффициентов кубического сплайна, построенного на двух точках.

Кубический сплайн вида будет иметь следующие коэффициенты:

$$\psi(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

$a = y_0$, $b = \frac{y1 - y0}{x1 - x0} * (x - x0)$, $c = 0$, $d = 0$.

2. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках.

Рассмотрим такой же кубический сплайн, как и в первом вопросе. Тогда его коэффициенту будут выглядеть так:

$a_1 = y_0$, $a_2 = y_1$,

$b_1 = \frac{y1 - y0}{x1 - x0} - (x1 - x0) \cdot (\frac{c2}{3})$, $b_2 = \frac{y2 - y1}{x2 - x1} - (x2 - x1) \cdot (\frac{2 \cdot c2}{3})$,

$c_1 = 0$, $c_2 = \frac{fi}{2 \cdot (x1 - x0 + x2 - x1)}$, где $fi = 3 \cdot (\frac{y2 - y1}{x2 - x1} - \frac{y1 - y0}{x1 - x0})$

$d_1 = c_2 / 3(x_1 - x_0)$, $d_2 = -c_2 / 3(x_1 - x_0)$.

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо $C_1 = C_2$.

Если $C_1 = C_2$, тогда в следующей формуле (положим $i = 1$), можно заменить C_i на C_{i+1} и получим что $\xi = 1$, а $\eta = 0$.

$$c_i = \xi_{i+1} c_{i+1} + \eta_{i+1},$$

4. Написать формулу для определения последнего коэффициента сплайна C_N, чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано $kC_{N-1} + mC_N = p$, где k, m, p – заданные числа.

$$CN = \frac{(3 \cdot (\frac{y_N - y_{(N-1)}}{x_N - x_{(N-1)}} - \frac{y_{(N-1)} - y_{(N-2)}}{x_{(N-1)} - x_{(N-2)}}) + ((x_{(N-1)} - x_{(N-2)})) \cdot (\frac{p}{k}))}{((-2) \cdot (x_{(N-1)} - x_{(N-2)} + x_N - x_{(N-1)}) + (x_{(N-1)} - x_{(N-2)}) \cdot (\frac{m}{k}))}$$

В знаменателе можно сократить x_{N-1}