|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_Программное обеспечение ЭВМ и информационные технологии\_\_\_\_\_\_\_\_

**Лабораторная работа № 3**

**Тема** Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

**Студент** Серова М. Н.

**Группа** ИУ7-45Б

**Оценка (баллы)** \_\_\_\_\_\_\_\_

**Преподаватель** Градов В. М.

*Москва,*

*2021 г.*

**Цель работы.** Получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

**Исходные данные.**

1. Таблица функции с количеством узлов N. Задать с помощью формулы y=x­2 в диапазоне [0; 10] с шагом 1.
2. Значение аргумента в первом интервале и в середине таблицы. Сравнить с точным значением.

**Код программы**

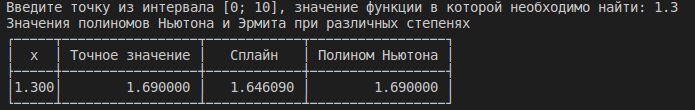
Код программы представлен на листингах 1-8

|  |
| --- |
| Листинг 1. polynoms.c  #include "../inc/spline.h"  error\_code **allocate\_and\_fill\_table**(table\_t \*table)  {      table->n = END\_NUM;      table->x = **calloc**(table->n, sizeof(double));      if (!table->x)      {  **printf**("Ошибка, не удалось выделить память\n");          return error\_memory;      }      table->y = **calloc**(table->n, sizeof(double));      if (!table->y)      {  **free**(table->x);  **printf**("Ошибка, не удалось выделить память\n");          return error\_memory;      }      for (int i = 0; i < table->n; i++)      {          table->x[i] = i;          table->y[i] = i \* i;      }      return no\_errors;  }  error\_code **allocte\_racing\_table**(table\_t \*racing)  {      racing->n = END\_NUM + 1;      racing->x = **calloc**(racing->n, sizeof(double));      if (!racing->x)      {  **printf**("Ошибка, не удалось выделить память\n");          return error\_memory;      }      racing->y = **calloc**(racing->n, sizeof(double));      if (!racing->y)      {  **free**(racing->x);  **printf**("Ошибка, не удалось выделить память\n");          return error\_memory;      }      racing->x[1] = 0;      racing->y[1] = 0;      return no\_errors;  }  double **h**(double \*x, int i)  {      return (x[i] - x[i - 1]);  }  double **f**(table\_t \*table, int i)  {      return 3 \* ((table->y[i] - table->y[i - 1]) / \  **h**(table->x, i) - (table->y[i - 1] - table->y[i - 2]) / **h**(table->x, i - 1));  }  void **get\_racing\_coeffs**(table\_t \*racing, table\_t \*table)  {      double hi = 0;      double hii = 0;      for (int i = 1; i < racing->n - 1; i++)      {          hi = **h**(table->x, i);          hii = **h**(table->x, i - 1);          racing->x[i + 1] = - hi / (hii \* racing->x[i] + \          2 \* (hii + hi));          racing->y[i + 1] = (**f**(table, i) - hii \* racing->y[i]) / \          (hii \* racing->x[i] + 2 \* (hii + hi));      }  }  void **free\_table**(table\_t \*table)  {      if (table->x)      {  **free**(table->x);          table->x = NULL;      }      if (table->y)      {  **free**(table->y);          table->y = NULL;      }      table->n = 0;  }  error\_code **get\_c\_coeff**(table\_t \*table, spline\_coeffs\_t \*coeffs, int i)  {      error\_code result = no\_errors;        table\_t racing;      result = **allocte\_racing\_table**(&racing);      if (result)          return result;    **get\_racing\_coeffs**(&racing, table);      coeffs->cii = 0;      coeffs->ci = 0;      for (int j = racing.n - 2; j >= i; j--)      {          coeffs->cii = coeffs->ci;          coeffs->ci = racing.x[j + 1] \* (coeffs->cii) + racing.y[j + 1];      }  **free\_table**(&racing);      return result;  }  error\_code **get\_coeffs**(table\_t \*table, spline\_coeffs\_t \*coeffs, int i)  {      error\_code result = **get\_c\_coeff**(table, coeffs, i);      if (result)          return result;      coeffs->a = table->y[i - 1];      if (i != table->n - 1)      {          coeffs->b = (table->y[i] - table->y[i - 1]) / **h**(table->x, i) - **h**(table->x, i) \* (coeffs->cii + 2 \* coeffs->ci) / 3;          coeffs->d = (coeffs->cii - coeffs->ci) / (3 \* **h**(table->x, i));      }      else      {          coeffs->b = (table->y[i] - table->y[i - 1]) / **h**(table->x, i) - **h**(table->x, i) \* 2 \* coeffs->ci / 3;          coeffs->d = -coeffs->ci / (3 \* **h**(table->x, i));      }      return result;  }  double **get\_spline\_result**(table\_t \*table, spline\_coeffs\_t \*coeffs, double x, int i)  {      double hi = x - table->x[i - 1];        return coeffs->a + coeffs->b \* hi + coeffs->ci \* hi \* hi + coeffs->d \* hi \* hi \* hi;  }  int **get\_interval\_index**(table\_t \*table, double x)  {      int index = 0;      for (int i = 0; (i < table->n - 1) && (x > table->x[i]); i++)      {          index = i;      }      return index + 1;  }  error\_code **get\_value\_spline**(double \*value, double x)  {      error\_code result = no\_errors;      table\_t table;      result = **allocate\_and\_fill\_table**(&table);      if (result)          return result;      int i = **get\_interval\_index**(&table, x);      spline\_coeffs\_t coeffs;      result = **get\_coeffs**(&table, &coeffs, i);      if (!result)      {          \*value = **get\_spline\_result**(&table, &coeffs, x, i);      }  **free\_table**(&table);      return result;  } |
| Листинг 2. spline.h  #ifndef **SPLINE\_H**  #define **SPLINE\_H**  #include "error\_code.h"  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  error\_code **get\_value\_spline**(double \*value, double x);  error\_code **allocate\_and\_fill\_table**(table\_t \*table);  void **free\_table**(table\_t \*table);  #endif |
| Листинг 3. scan.c  #include "../inc/scan.h"  error\_code **scan\_number**(double \*x)  {      error\_code result = no\_errors;  **printf**("Введите точку из интервала [0; %d], значение функции в которой необходимо найти: ", END\_NUM - 1);      int check = **scanf**("%lf", x);      if (check != 1)      {  **printf**("Точка должна быть вещественным числом\n");          result = error\_input;      }      if (\*x < 0 || \*x > END\_NUM - 1)      {  **printf**("Ошибка, точка должна принадлежать интервалу [0; %d]\n", END\_NUM - 1);          result = error\_input;      }      return result;  } |
| Листинг 4. scan.h  #ifndef **SCAN\_H**  #define **SCAN\_H**  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "error\_code.h"  error\_code **scan\_number**(double \*x);  #endif |
| Листинг 5. polynomNewton.c  #include "../inc/polynomNewton.h"  int **get\_start\_index**(table\_t \*table, double x, int n)  {      int index = 0;      for (int i = 0; i < table->n - 1; i++)      {          if ((table->x[i] <= x && table->x[i + 1] >= x) || (table->x[i] >= x && table->x[i + 1] <= x))              index = i;      }      index -= n / 2 - 1;      if (index + n > table->n)          index = table->n - n - 1;      return (index < 0) ? 0 : index;  }  void **swap**(double \*\*first, double \*\*second)  {      double \*tmp = \*first;      \*first = \*second;      \*second = tmp;  }  error\_code **find\_value\_polynomial**(double x, int n, double \*p\_x)  {      error\_code result = no\_errors;      n += 1;      int start = 0;      double \*tmp = NULL;      table\_t table =      {          .x = NULL,          .y = NULL,      };      result = **allocate\_and\_fill\_table**(&table);      if (!result)      {          start = **get\_start\_index**(&table, x, n);          tmp = **calloc**(table.n, sizeof(double));          if (!tmp)              result = error\_memory;      }      if (!result)      {          \*p\_x = table.y[start];  **memmove**(tmp, table.y, table.n \* sizeof(double));          for (int i = start + 1; i < n + start; i++)          {              for (int j = start; j < n - i + start + start; j++)              {                    tmp[j] = (tmp[j] - tmp[j + 1]) / (table.x[j] - table.x[j + i - start]);              }              double prom = tmp[start];              for (int j = start; j < i; j++)              {                  prom \*= (x - table.x[j]);              }              \*p\_x += prom;          }      }  **free\_table**(&table);      if (tmp)  **free**(tmp);      return result;  } |
| Листинг 6. polynomNewton.h  #ifndef **POLYNOMNEWTON\_H**  #define **POLYNOMNEWTON\_H**  #include "error\_code.h"  #include "spline.h"  #include <string.h>  error\_code **find\_value\_polynomial**(double x, int n, double \*p\_x);  #endif |
| Листинг 7. error\_code.h  #ifndef **ERROR\_CODE\_H**  #define **ERROR\_CODE\_H**  #define **END\_NUM** 11  typedef enum  {      no\_errors,      error\_input,      error\_memory,  } **error\_code**;  typedef struct  {      double \*x;      double \*y;      int n;  } **table\_t**;  typedef struct  {      double a;      double b;      double ci;      double cii;      double d;  } **spline\_coeffs\_t**;  #endif |
| Листинг 8. main.c  #include "../inc/scan.h"  #include "../inc/spline.h"  #include "../inc/polynomNewton.h"  int **main**(int argc, char \*\*argv)  {      error\_code result = no\_errors;      double x = 0;      double value = 0;      double newton = 0;      result = **scan\_number**(&x);      if (!result)      {          result = **get\_value\_spline**(&value, x);          if (!result)              result = **find\_value\_polynomial**(x, 3, &newton);      }      if (!result)      {  **printf**("Значения полиномов Ньютона и Эрмита при различных степенях\n"                 "┌─────┬─────────────────┬────────────┬─────────────────┐\n"                 "│  x  │ Точное значение │   Сплайн   │ Полином Ньютона │\n"                 "├─────┼─────────────────┼────────────┼─────────────────┤\n");  **printf**("│%5.3lf│ %15.6lf │ %10.6lf │ %15.6lf │\n", x, x \* x, value, newton);  **printf**("└─────┴─────────────────┴────────────┴─────────────────┘\n");      }      return result;  } |

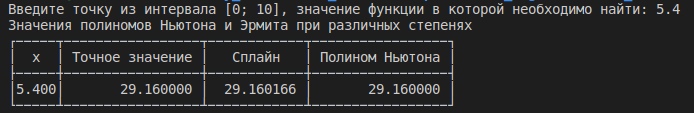
**Результаты работы.**

1. Значения y(x).
2. Сравнить результаты интерполяции кубическим сплайном и полиномом Ньютона 3-ей степени.

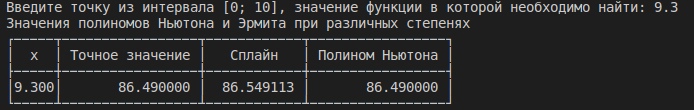
Результат для значения аргумента в первом интервале:



Результат для значения аргумента в середине таблицы:



Результат для значения аргумента в конце таблицы:



**Вопросы при защите лабораторной работы.**

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

Рассмотрим 2 точки (x1, y1) и (x2, y2) такие, что x1 и x2 различны. Таким образом, получим сплайн, рассматриваемый только на одном интервале, его вид:

Учтем, что в узлах значения многочлена и интерполируемой функции совпадают:

Подставим значения в сплайн:

Для получения значения четырех переменных необходимо добавить еще два условия (два перечислены выше). Для этого приравняем вторые производные к нулю (будем считать, что первые и вторые производные сплайна непрерывны):

Вычислим значения:

В итоге, получаем следующую систему уравнений:

Из последних двух уравнений получаем

Подставим найденные значения в сплайн и получим

Итоговое решение выглядит следующим образом:

Получаем результирующую формулу для кубического сплайна, построенного на двух точках:

1. Вычислить все условия для определения коэффициентов сплайна, построенного на трех точках

Сплайн, построенный на трех точках, будет состоять из двух функций и , в каждой из которых – 4 неизвестных коэффициента. Итого 8 неизвестных.

Первые 4 условия получим из условия значений сплайна в узлах:

Следующие два условия вытекают из условия для значений производных:

Недостающие 2 условия определим следующим образом:

1. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо c1=c2.

Пусть c1 = c2 = с, причем c1 = 0 (из условия равенства нулю второй производной на концах участка интерполирования). Тогда

Отсюда

1. Написать формулу для определения последнего коэффициента сплайна CN, чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано kCN-1+mCN=p, где k, m и p – заданные числа.