|  |  |
| --- | --- |
| 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 № 1**

**Тема** Построение и программная реализация алгоритма полиномиальной интерполяции табличных функций.

**Студент** Серова М. Н.

**Группа** ИУ7-45Б

**Оценка (баллы)** \_\_\_\_\_\_\_\_

**Преподаватель** Градов В. М.

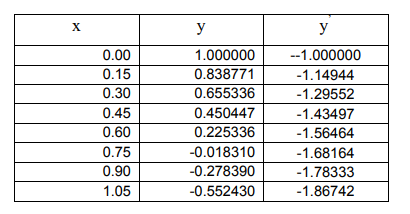
*Москва,*

*2021 г.*

**Цель работы.** Получение навыков построения алгоритма интерполяции таблично заданных функций полиномами Ньютона и Эрмита.

**Исходные данные.**

1. Таблица функции и её производных



1. Степень аппроксимирующего полинома – n.
2. Значение аргумента, для которого выполняется интерполяция.

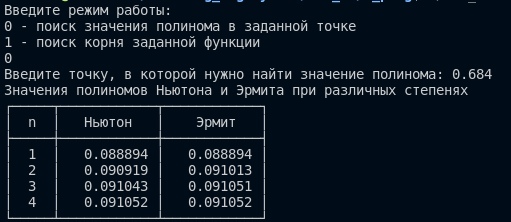
**Код программы**

Код программы представлен на листингах 1-3

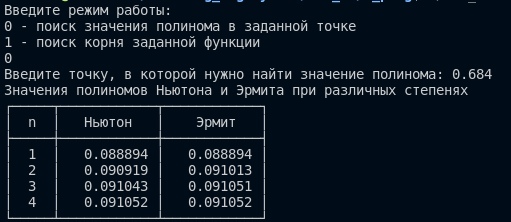
|  |
| --- |
| Листинг 1. polynoms.c  #include "../inc/polynoms.h"  void **free\_table**(**table\_t** \*table)  {      if (table->x)  **free**(table->x);      if (table->y)  **free**(table->y);      if (table->y\_point)  **free**(table->y\_point);  }  **error\_code** **get\_table**(**table\_t** \*table)  {  **error\_code** result = no\_errors;      table->x = **calloc**(**LENGTH** \* table->type, sizeof(double));      if (!table->x)      {          result = error\_memory;      }      else      {          table->y = **calloc**(**LENGTH** \* table->type, sizeof(double));          if (!table->y)          {              result = error\_memory;          }          else          {              table->y\_point = **calloc**(**LENGTH**, sizeof(double));              if (!table->y\_point)              {                  result = error\_memory;              }          }      }      if (!result)      {          double cnt = 0;          for (int i = 0; i < **LENGTH** \* table->type; i++)          {              table->x[i] = cnt;              if ((table->type == Hermit && i % 2 == 1) || table->type == Newton)                  cnt += 0.15;          }          int i = 0;          for (int j = 0; j < table->type; j++)              table->y[i + j] = 1;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = 0.838771;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = 0.655336;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = 0.450447;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = 0.225336;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = -0.018310;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = -0.278390;          i += table->type;          for (int j = 0; j < table->type; j++)              table->y[i + j] = -0.552430;          table->y\_point[0] = -1;          table->y\_point[1] = -1.14944;          table->y\_point[2] = -1.29552;          table->y\_point[3] = -1.43497;          table->y\_point[4] = -1.56464;          table->y\_point[5] = -1.68164;          table->y\_point[6] = -1.78333;          table->y\_point[7] = -1.86742;      }      return result;  }  int **get\_start\_index**(**table\_t** \*table, double x, int n)  {      int index = 0;      for (int i = 0; i < **LENGTH** \* table->type - 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 > **LENGTH** \* table->type)          index = **LENGTH** \* table->type - n - 1;      return (index < 0) ? 0 : index;  }  void **swap**(double \*\*first, double \*\*second)  {      double \*tmp = \*first;      \*first = \*second;      \*second = tmp;  }  double **find\_value\_polynomial**(double x, int n, **error\_code** \*result, **mode** current, int polynome\_type)  {      double p\_x = 0;      n += 1;      int start = 0;      double \*tmp = NULL;  **table\_t** table =      {          .x = NULL,          .y = NULL,          .y\_point = NULL,          .type = polynome\_type,      };      \*result = **get\_table**(&table);      if (!\*result)      {          if (current == root)          {  **swap**(&table.x, &table.y);          }          start = **get\_start\_index**(&table, x, n);          tmp = **calloc**(**LENGTH** \* table.type, sizeof(double));          if (!tmp)              \*result = error\_memory;      }      if (!\*result)      {          p\_x = table.y[start];  **memmove**(tmp, table.y, **LENGTH** \* table.type \* sizeof(double));          for (int i = start + 1; i < n + start; i++)          {              for (int j = start; j < n - i + start + start; j++)              {                  if (table.type == Hermit && **fabs**(tmp[j] - tmp[j + 1]) < **EPS**)                  {                      tmp[j] = table.y\_point[j / 2];                  }                  else                  {                      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 p\_x;  }  **error\_code** **scan\_numbers**(double \*x, int \*curr\_mode)  {  **error\_code** result = no\_errors;  **printf**("Введите режим работы:\n"      "0 - поиск значения полинома в заданной точке\n"      "1 - поиск корня заданной функции\n");      int check = **scanf**("%d", curr\_mode);      if (check != 1 || \*curr\_mode < 0 || \*curr\_mode > 1)      {          result = error\_input;  **printf**("Ошибка, режим работы - целое число (0 или 1)\n");      }      if (!result && \*curr\_mode == normal)      {  **printf**("Введите точку, в которой нужно найти значение полинома: ");          check = **scanf**("%lf", x);          if (check != 1)          {  **printf**("Ошибка, точка для поиска значения полинома должна быть вещественным числом\n");              result = error\_input;          }      }      return result;  } |
| Листинг 2. polynoms.h  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <math.h>  #define **LENGTH** 8  #define **EPS** 1e-8  #define **START\_POW** 1  #define **END\_POW** 4  typedef struct  {      double \*x;      double \*y;      double \*y\_point;      int type;  } **table\_t**;  typedef enum  {      normal,      root,  } **mode**;  typedef enum  {      no\_errors,      error\_memory,      error\_input,  } **error\_code**;  typedef enum  {      Newton = 1,      Hermit = 2,  } **type\_table**;  **error\_code** **scan\_numbers**(double \*x, int \*curr\_mode);  double **find\_value\_polynomial**(double x, int n, **error\_code** \*result, **mode** current, int polynome\_type); |
| Листинг 3. main.c  #include "../inc/polynoms.h"  int **main**(void)  {  **error\_code** result = no\_errors;      double x = 0;      int curr\_mode = normal;      result = **scan\_numbers**(&x, &curr\_mode);      if (!result && curr\_mode == normal)      {  **printf**("Значения полиномов Ньютона и Эрмита при различных степенях\n"                 "┌─────┬────────────┬────────────┐\n"                 "│  n  │   Ньютон   │    Эрмит   │\n"                 "├─────┼────────────┼────────────┤\n");          for (int i = **START\_POW**; i <= **END\_POW** && !result; i++)          {  **printf**("│  %d  │ %10.6f │ %10.6f │\n", i, \  **find\_value\_polynomial**(x, i, &result, curr\_mode, Newton),  **find\_value\_polynomial**(x, i, &result, curr\_mode, Hermit));          }  **printf**("└─────┴────────────┴────────────┘\n");      }      else if (!result)      {  **printf**("Корень заданной табличной функции при различных степенях полинома Ньютона при обратной интерполяции\n"                 "┌─────┬────────────┐\n"                 "│  n  │   Корень   │\n"                 "├─────┼────────────┤\n");          for (int i = **START\_POW**; i <= **END\_POW** && !result; i++)          {  **printf**("│  %d  │ %10.6f │\n", i, \  **find\_value\_polynomial**(0, i, &result, curr\_mode, Newton));          }  **printf**("└─────┴────────────┘\n");        }      return result;  } |

**Результаты работы.**

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n= 1, 2, 3 и 4 при фиксированном x



1. Найти корень заданной выше табличной функции с помощью обратной интерполяции, используя полином Ньютона



**Вопросы при защите лабораторной работы.**

1. Будет ли работать программа при степени полинома n=0?

При n=0 программа будет работать, однако выдаваемый ответ будет совпадать со значением полинома в начале выбранного для интерполяции отрезка.

1. Как практически оценить погрешность интерполяции? Почему сложно применить для этих целей теоретическую оценку?

Для этого нужно знать, какая именно функция представлена в виде табличных значений, и сравнить истинное значение с полученным при интерполяции. Также погрешность можно оценить посредством увеличения степени полинома и оценки изменения его значения. Теоретическую оценку применить сложно, поскольку она подразумевает необходимость наличия значений производных (вплоть до n + 1 производной).

1. Если в двух точках заданы значения функции и ее первых производных, то полином какой минимальной степени может быть построен на этих точках?

Минимальная степень такого полинома – 3, поскольку задано 4 условия (узла)

1. В каком месте алгоритма построения полинома существенна информация об упорядоченности аргумента функции (возрастает, убывает)?

Эта информация важна при выборе отрезка для интерполяции. Если аргумент функции неупорядочен, выбор соответствующего отрезка будет нетривиальной задачей.

1. Что такое выравнивающие переменные и как их применить для повышения точности интерполяции?

Выравнивающие переменные – функции от переменных x и y такие, что в этих переменных график максимально близок к прямой на отдельных участках, они используются, когда имеется быстроменяющаяся функция, представление которой в виде таблицы требует больших объемов памяти. Тогда сначала интерполяцию производят в новых переменных, а затем обратным интерполированием находят значение y(x).