|  |  |
| --- | --- |
| 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:** Золотухин А. В.

**Группа:** ИУ7-44Б

**Оценка (баллы):** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Преподаватель:** Градов В.М.

*Москва*

*2022 г*

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

**Исходные данные**

1. Таблица функции и её производных

|  |  |  |
| --- | --- | --- |
| X | Y | Y` |
| 0 | 1 | -1 |
| 0.15 | 0.838771 | -1.14944 |
| 0.30 | 0.655336 | -1.29552 |
| 0.45 | 0.450447 | -1.43497 |
| 0.60 | 0.225336 | -1.56464 |
| 0.75 | -0.018310 | -1.68164 |
| 0.90 | -0.278390 | -1.78333 |
| 1.05 | -0.552430 | -1.86742 |

2. Степень аппроксимирующего полинома — n.

3. Значение аргумента, для которого выполняется интерполяция.

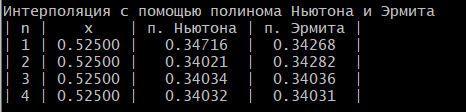
**Код программы**

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #define MAX\_POINTS\_COUNT 50  #define COL\_COUNT 3  int parse\_table(double matr[][COL\_COUNT], int \*row, FILE \*f)  {      if (fscanf(f, "%d", row) != 1 && \*row > MAX\_POINTS\_COUNT)          return EXIT\_FAILURE;      for (int i = 0; i < \*row; i++)          for (int j = 0; j < COL\_COUNT; j++)              if (fscanf(f, "%lf", &(matr[i][j])) != 1)                  return EXIT\_FAILURE;      return EXIT\_SUCCESS;  }  void split\_matr(double matr[][COL\_COUNT], int row, double x[], double y[], double d[])  {      for (int i = 0; i < row; i++)      {          x[i] = matr[i][0];          y[i] = matr[i][1];          d[i] = matr[i][2];      }  }  double input\_x(void)  {      printf("Enter X: \n");      int flag = 0;      double x;      while (flag == 0)          if (scanf("%lf", &x) == 1)              flag = 1;          else              printf("Some error! Try again");      return x;  }  int strip\_arr(double src[], double dst[], int i\_b, int i\_e)  {      int dst\_len = 0;      for (int i = i\_b; i < i\_e; i++)          dst[dst\_len++] = src[i];      return dst\_len;  }  void find\_x0\_xn(double arr[], int row, int power, double arg, int \*ind\_x0, int \*ind\_xn)  {      int index\_x = 0;      while (arg > arr[index\_x++]);      \*ind\_x0 = index\_x - power / 2 - 1;      \*ind\_xn = index\_x + (power / 2) + (power % 2) - 1;      if (\*ind\_xn > row - 1)      {          \*ind\_x0 -= \*ind\_xn - row + 1;          \*ind\_xn = row - 1;      }      else if (\*ind\_x0 < 0)      {          \*ind\_xn -= \*ind\_x0;          \*ind\_x0 = 0;      }  }  void div\_diff(double x[], double y[], int node, double \*poly[], int \*poly\_n)  {      \*poly\_n = 0;      for (int i = 0; i < node; i++)          poly[(\*poly\_n)++] = calloc(node + 1, sizeof(double));      for (int i = 0; i < node; i++)      {          poly[i][0] = x[i];          poly[i][1] = y[i];      }      int i = 2;      int new\_node = node - 1;      while (i < (node + 1))      {          int j = 0;          while (j < new\_node)          {              poly[j][i] = (poly[j + 1][i - 1] - poly[j][i - 1]) / (poly[i - 1][0] - poly[0][0]);              j++;          }          i++;          new\_node--;      }  }  double polinom\_n(double x[], double y[], int node, double arg)  {      double \*poly[MAX\_POINTS\_COUNT];      int poly\_n;      div\_diff(x, y, node, poly, &poly\_n);      double arg\_y = poly[0][1];      int i = 2;      while (i < node + 1)      {          int j = 0;          double p = 1;          while (j < i - 1)              p \*= (arg - poly[j++][0]);          arg\_y += poly[0][i++] \* p;      }      return arg\_y;  }  void hermite\_interpolate(int node, double x[], double y[], double d[], int n, double \*poly[], int \*poly\_n)  {      \*poly\_n = 0;      for (int i = 0; i < 2 \* n; i++)          poly[(\*poly\_n)++] = calloc(2 \* node + 3, sizeof(double));      int i = 0;      for (int j = 0; j < n; j++)      {          poly[i][0] = x[j];          poly[i][1] = y[j];          poly[i++][2] = d[j];          poly[i][0] = x[j];          poly[i++][1] = y[j];      }      i = 2;      for (int j = 0; j < (\*poly\_n) - 1; j++)          if (j % 2 == 1)              poly[j][i] = (poly[j][1] - poly[j + 1][1]) / (poly[j][0] - poly[j + 1][0]);      i = 3;      int new\_node = node - 2;      while (i < \*poly\_n)      {          int j = 0;          while (j < new\_node)          {              poly[j][i] = (poly[j + 1][i - 1] - poly[j][i - 1]) / (poly[i - 1][0] - poly[0][0]);              j++;          }          i++;          new\_node--;      }  }  double polynom\_h(double \*poly[], int node, double arg)  {      double y = poly[0][1];      int i = 2;      while (i < node + 2)      {          int j = 0;          double p = 1;          while (j < i - 1)              p \*= (arg - poly[j++][0]);          y += poly[0][i++] \* p;      }      return y;  }  void bubblesort(double a[], int n, double b[])  {      int f = 1;      while (f)      {          f = 0;          for (int i = 0; i < n - 1; i++)              if (a[i] > a[i + 1])              {                  f = 1;                  double tmp = a[i];                  a[i] = a[i + 1];                  a[i + 1] = tmp;                  tmp = b[i];                  b[i] = b[i + 1];                  b[i + 1] = tmp;              }      }  }  int main(int argc, char \*\*argv)  {      setbuf(stdout, NULL);      if (argc != 2)          return EXIT\_FAILURE;      FILE \*filein = fopen(argv[1], "r");      if (filein == NULL)          return EXIT\_FAILURE;      double a[MAX\_POINTS\_COUNT][COL\_COUNT];      int row;      double x = input\_x();      parse\_table(a, &row, filein);      double coords\_x[MAX\_POINTS\_COUNT], coords\_y[MAX\_POINTS\_COUNT], coords\_d[MAX\_POINTS\_COUNT];      split\_matr(a, row, coords\_x, coords\_y, coords\_d);      for (int i = 0; i < row; i++)      {          for (int j = 0; j < COL\_COUNT; j++)              printf("%lf ", a[i][j]);          printf("\n");      }      printf("\nИнтерполяция с помощью полинома Ньютона и Эрмита\n"            "| n |    x    | п. Ньютона | п. Эрмита |\n");      for (int n = 1; n < 5; n++)      {          printf("| %d | %.5lf |", n, x);          int flag = 0;          for (int i = 0; i < row; i++)              if (fabs(x - a[i][0]) < 1e-8)              {                  printf("   %.5lf  |", a[i][1]);                  flag = 1;              }          if (!flag)          {              int x0, xn;              find\_x0\_xn(coords\_x, row, n, x, &x0, &xn);              double ax[MAX\_POINTS\_COUNT], ay[MAX\_POINTS\_COUNT], ad[MAX\_POINTS\_COUNT];              int ax\_n = strip\_arr(coords\_x, ax, x0, xn + 1);              strip\_arr(coords\_y, ay, x0, xn + 1);              strip\_arr(coords\_d, ad, x0, xn + 1);              if (ax\_n)              {                  double my\_root = polinom\_n(ax, ay, n + 1, x);                  printf("   %.5lf  |", my\_root);              }              double \*poly[2 \* MAX\_POINTS\_COUNT];              int poly\_n;              hermite\_interpolate(n, ax, ay, ad, ax\_n, poly, &poly\_n);              double my\_root2 = polynom\_h(poly, n, x);              printf("  %.5lf  |\n", my\_root2);          }      }      printf("\nОбратная интерполяция с помощью полинома Ньютона\n"            "| n |     x     |   Корень   |\n");      for (int n = 1; n < 5; n++)      {          printf("| %d |  %.5lf  |", n, 0.0);          int flag = 0;          for (int i = 0; i < row; i++)              if (fabs(a[i][1]) < 1e-8)              {                  printf("  %lf |\n", a[i][1]);                  flag = 1;              }          if (!flag)          {              int y0, yn;              bubblesort(coords\_y, row, coords\_x);              find\_x0\_xn(coords\_y, row, n, 0.0, &y0, &yn);              double ax[MAX\_POINTS\_COUNT], ay[MAX\_POINTS\_COUNT];              int ay\_n = strip\_arr(coords\_y, ay, y0, yn + 1);              strip\_arr(coords\_x, ax, y0, yn + 1);              if (ay\_n)              {                  double my\_root = polinom\_n(ay, ax, n + 1, 0.0);                  printf("   %.5lf  |\n", my\_root);              }          }      }      fclose(filein);      return EXIT\_SUCCESS;  } |

**Результаты работы**

1. Значения y(x) при степенях полиномов Ньютона и Эрмита n= 1, 2, 3 и 4 при фиксированном x, например, x=0.525 (середина интервала 0.45- 0.60). Результаты свести в таблицу для сравнения полиномов.

(Вывод программы)

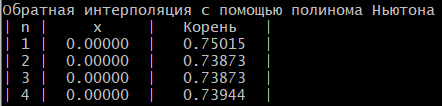


|  |  |  |  |
| --- | --- | --- | --- |
| n | x | п. Ньютона | п. Эрмита |
| 1 | 0.525 | 0.33789 | 0.34282 |
| 2 | 0.525 | 0.34042 | 0.34282 |
| 3 | 0.525 | 0.34031 | 0.34036 |
| 4 | 0.525 | 0.34032 | 0.34031 |

*(Таблица в исходных данных составлена по функции y(x) = cos(x) - x; Точное значение в точке x = 0.525 равно 0.3403239416229412)*

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

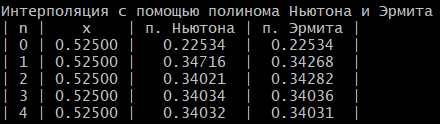
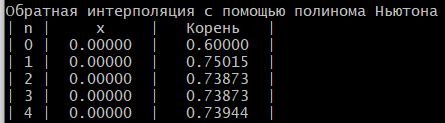
(Вывод программы)



|  |  |  |
| --- | --- | --- |
| n | x | Корень |
| 1 | 0 | 0.73873 |
| 2 | 0 | 0.73944 |
| 3 | 0 | 0.73944 |
| 4 | 0 | 0.74762 |

**Вопросы при защите лабораторной работы**

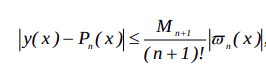
***1. Будет ли работать программа при степени полинома n=0?***

Да, работать программа с такой степенью будет, как расчет интерполяции с помощью полинома Ньютона, так и с помощью полинома Эрмита. Но точность при такой конфигурации будет низкой.   


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

Практически оценить погрешность интерполяции можно при помощи оценки первого отброшенного члена в полиноме Ньютона. При этом в полиноме остаются члены, которые больше заданной погрешности расчетов.

Теоретическую погрешность многочлена Ньютона можно оценить с помощью формулы (где используются производные данной функции):

, где  - максимальное значение производной интерполируемой функции, а также 

Именно поэтому теоретическую погрешность сложно оценить.

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

При данном условии можно построить полиномы Эрмита 0, 1, 2 и 3 степени а полиномы Ньютона - 0 и 1 степени.  
Минимальная степень равна 0.

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

Информация об упорядоченности аргумента функции существенна при выборе приближенного интервала значений (из (n+1) узлов, которые по возможности расположены симметрично относительно заданного аргумента).

Если аргумент будет неупорядоченным, то значение функции получится с низкой точностью или совсем неверным.

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

Если функция, а точнее ее разделенные разности, значительно меняются на нескольких интервалах, то интерполяция обобщенным многочленом обычно не будет точной для дифференцирования данной функции. Поэтому для таких функций используется квазилинейная интерполяция, которая производится при помощи выравнивающих переменных. То есть выравнивающие переменные используются для того, чтобы повысить точность вычисления производной функции.