Санкт-Петербургский Государственный Университет
Прикладная математика и информатика

Отчёт по вычислительному практикуму 3
Задача обратного интерполирования. Нахождение производных таблично-заданной функции по формулам численного дифференцирования

Выполнил:
Яковлев Денис Михайлович
Группа 21.Б06-мм
st095998@student.spbu.ru

Под руководством Алцыбеева Глеба Олеговича
Преподавателя и ассистента по дисциплине "Вычислительный практикум"

17 Марта 2023 г.

Содержание

1 Введение

2 Постановка задач

2.1 (Задача №3.1) Задача обратного интерполирования

2.1.1 Подготовительный этап

Вывести на печать таблицу из $(m+1)$ значения функции f в равноотстоящих с шагом $h = \frac{(b-a)}{m}$ точках (узлах) $x_i = a + ih$, где $i = 0, 1, ..., m$. Узлы x_i — точки деления отрезка $[a, b]$ на m частей.
Здесь число значений в таблице $m+1, a, b$ — **параметры задачи**; формула для непрерывной функции f, значениями которой заполняется таблица, дана в варианте тестовой задачи.
Параметры задачи запрашивать у пользователя, вводить с клавиатуры.

2.1.2 Решение задачи обратного интерполирования

Дана таблично-заданная функция (смотри таблицу, созданную на подготовительном этапе). Найти значение/значения аргумента/аргументов (задача может иметь не единственное решение!), при котором данная таблично−заданная функция принимает значение F, здесь F — параметр задачи.

2.1.2.1 Первый способ решения

Пусть таблично-заданная функция, для которой решается задача, строго монотонна (предполагается, что функция f, таблица которой дана в задаче, на рассматриваемом участке — это строго монотонная и непрерывная функция, то у нее существует обратная функция f^{-1}, которая также строго монотонна и непрерывна). Тогда задача обратного интерполирования может быть сведена к задаче поиска значения $f^{-1}(F)$ для таблично-заданной функции f^{-1}(при этом следует поменять местами столбцы исходной таблицы и далее трактовать значения $f(x_i)$ как аргументы для f^{-1}). Таким образом, имеем задачу алгебраического интерполирования для таблично-заданной функции f^{-1}, где F — точка интерполирования. Теперь, если построить интерполяционный многочлен Q_n по таблице значений, то решением задачи будет значение $Q_n(F) \approx f^{-1}(F)$.
Степень интерполяционного многочлена n — параметр задачи $(n \leq m)$ — запросить у пользователя. При нахождении значения $Q_n(F)$ использовать программу из Задания №2 (представление в форме Лагранжа или Ньютона – неважно).
Результатом решения задачи обратного интерполирования 1 способом является значение $X = Q_n(F)$.
ПРОВЕРКА: В тестовой задаче всегда можно посчитать модуль невязки $r_n(X) = |f(X) - F|$

2.1.2.2 Второй способ решения

Если мы не располагаем информацией, что на рассматриваемом участке таблицы функция строго монотонна и непрерывна, и, следовательно, не полномочны «переворачивать таблицу», то возможно следующее решение. Также этот способ решения можно применять, если первый способ возможен, но не дал хороший результат (например, если обратная функция плохо приближается многочленом).
Результатом решения задачи обратного интерполирования 2 способом будет(ут) корень(ни)

уравнения $P_n(x) = F$, где $P_n(x)$ – интерполяционный полином функции $f(x)$. При построении интерполяционного многочлена $P_n(x)$ можно использовать программу из ЛР №2. Алгебраическое уравнение решить методом секущих или методом бисекции с точностью ε (смотри ЛР №1).
ПРОВЕРКА: В тестовой задаче всегда можно посчитать модуль невязки $r_n(x) = |f(x) - F|$ для каждого при- ближенного решения.

2.1.2.3 Уточнение к решению тестовой задачи

В задаче обратного интерполирования взять формулу для функции из своего варианта Задания №2; $a = 0, b = 1, m = 10, \varepsilon = 10^{-8}$. Предусмотреть возможность ввода новых значений параметров F, n *и* ε.

2.2 (Задача №3.2) Нахождение производных таблично-заданной функции по формулам численного дифференцирования

2.2.1 Подготовительный этап

Вывести на печать таблицу из $(m + 1)$ значения функции f в равноотстоящих с шагом h точках $x_i = a + ih$, где $i = 0, 1, ..., m$. Рассматривать функцию $f(x) = e^{1,5kx}$, где $k = ($*(номер Вашего варианта по mod 5)* $+ 1)$. Здесь число значений в таблице $m + 1, a, h > 0$ — параметры задачи; Параметры задачи запрашивать у пользователя, вводить с клавиатуры.

2.2.2 Решение задачи численного дифференцирования

Для таблично-заданной функции f, найти значение ее первой и второй производной в узлах x_i таблицы. Для этого воспользоваться известными простейшими формулами численного дифференцирования, имеющими погрешность, порядка $O(h^2)$.

Вывести на печать таблицу вида:

x_i	$f(x_i)$	$f'(x_i)_{ЧД}$	$\lvert f'(x_i)_T - f'(x_i)_{ЧД}\rvert$	$f''(x_i)_{ЧД}$	$\lvert f''(x_i)_T - f''(x_i)_{ЧД}\rvert$

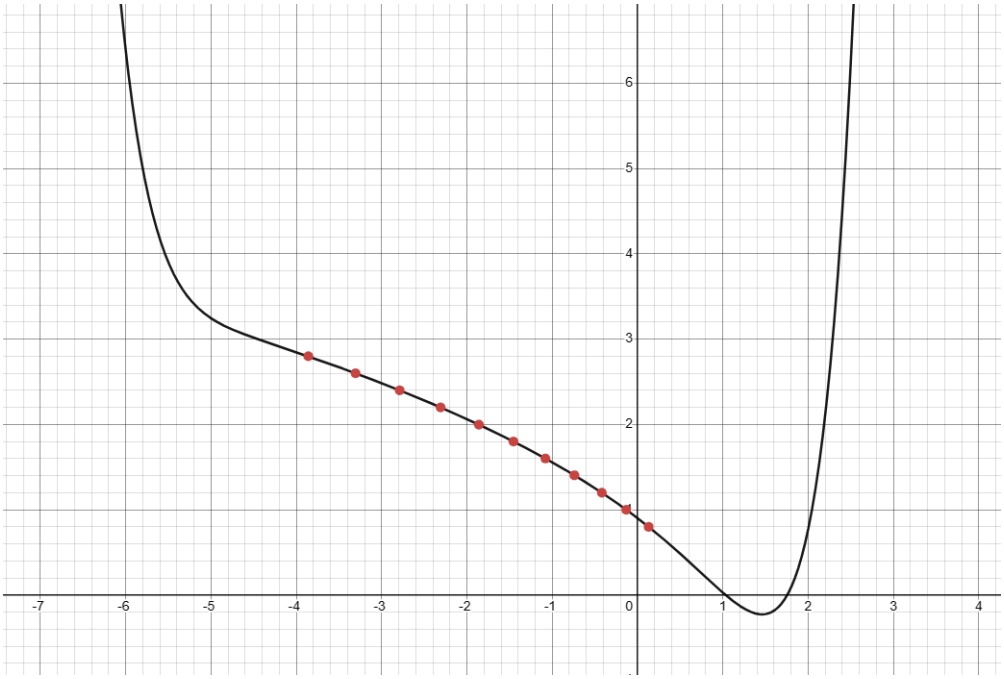
3 Ход работы

3.1 Задача №3.1

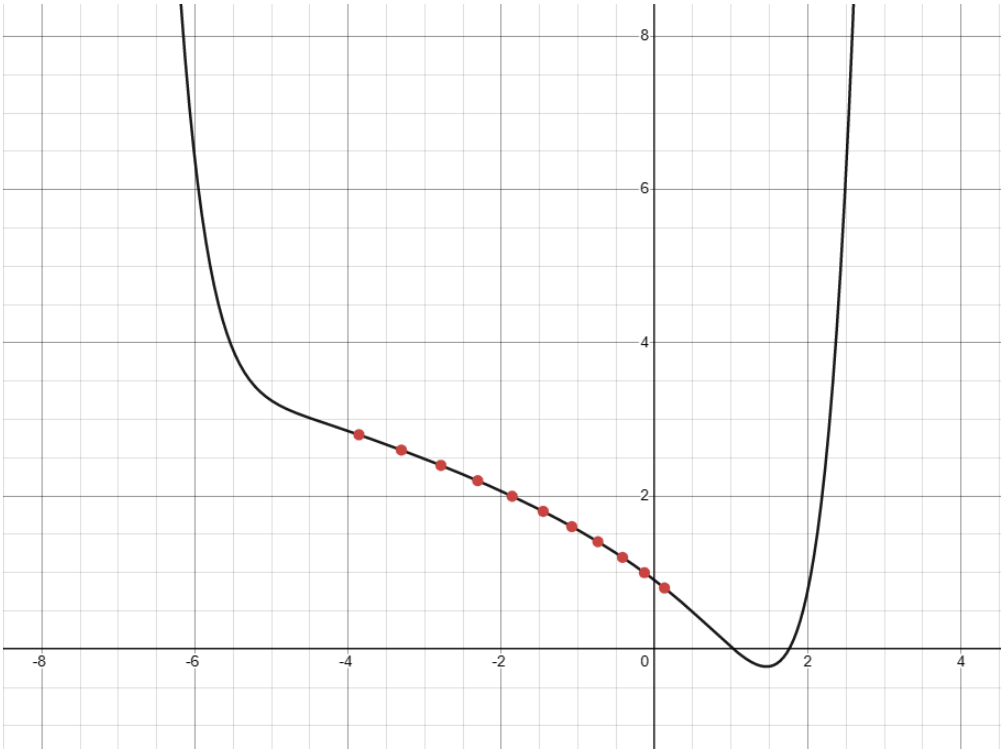
3.1.1 Первый способ решения

На вход запрашиваются следующие параметры: начало отрезка a, конец отрезка b, число аргументов $m+1$ и степень интерполяционного многочлена n ($n \leq m$), точка интерполяции x. В качестве входных параметров:

- функция $f(x) = e^{-x} - \frac{x^2}{2}$;
- $a = 0$ - начало отрезка;
- $b = 5$ - конец отрезка;
- $(m+1) = 26$ - число аргументов;
- $n = 10$ - степень интерполяционного многочлена;
- $x = 2;\ x = 8$ - две точки интерполяции;



(a) График интерполяционного многочлена Ньютона. Значения в узлах и точках интерполирования выделены красным цветом.



(b) График интерполяционного многочлена Лагранжа. Значения в узлах и точках интерполирования выделены красным цветом.

Ссылка на репозиторий: `https://github.com/DeMiYak/NumericalMethods`

3.1.2 Вывод

```
Вариант 7. Функция f(x) = eps(-x) - x^2/2

Выберите способ решения задачи №3.1
1. Первый (интерполяционный)
2. Второй (через корни уравнения)
Или любое другое число, чтобы выйти
1

Первый способ решения:
Начало отрезка: 0
Конец отрезка: 5
Число аргументов: 26
x                       f(x)
0.0000000000000000      1.0000000000000000
0.2000000000000000      0.7987307530779818
0.4000000000000000      0.5903200460356393
0.6000000000000001      0.3688116360940263
0.8000000000000000      0.1293289641172215
1.0000000000000000     -0.1321205588285577
1.2000000000000000     -0.4188057880877978
1.3999999999999999     -0.7334030360583934
1.5999999999999999     -1.0781034820053443
1.7999999999999998     -1.4547011117784132
1.9999999999999998     -1.8646647167633867
2.1999999999999997     -2.3091968416376654
2.3999999999999999     -2.7892820467105874
2.6000000000000001     -3.3057264217856663
2.8000000000000003     -3.8591899373747829
3.0000000000000004     -4.4502129316321382
3.2000000000000006     -5.0792377960216353
3.4000000000000008     -5.7466267300396767
3.6000000000000010     -6.4526762775527109
3.8000000000000012     -7.1976292281438390
4.0000000000000009     -7.9816843611112693
4.2000000000000011     -8.8050044231795255
4.4000000000000012     -9.6677226600969366
4.6000000000000014    -10.5699481642553739
4.8000000000000016    -11.5117702529509884
5.0000000000000018    -12.4932620530009242

Введите точку интерполяции f(x): -2

f(x)                            x
```

```
1.0000000000000000      0.0000000000000000
0.7987307530779818      0.2000000000000000
0.5903200460356393      0.4000000000000000
0.3688116360940263      0.6000000000000001
0.1293289641172215      0.8000000000000000
-0.1321205588285577      1.0000000000000000
-0.4188057880877978      1.2000000000000000
-0.7334030360583934      1.3999999999999999
-1.0781034820053443      1.5999999999999999
-1.4547011117784132      1.7999999999999998
-1.8646647167633867      1.9999999999999998
-2.3091968416376654      2.1999999999999997
-2.7892820467105874      2.3999999999999999
-3.3057264217856663      2.6000000000000001
-3.8591899373747829      2.8000000000000003
-4.4502129316321382      3.0000000000000004
-5.0792377960216353      3.2000000000000006
-5.7466267300396767      3.4000000000000008
-6.4526762775527109      3.6000000000000010
-7.1976292281438390      3.8000000000000012
-7.9816843611112693      4.0000000000000009
-8.8050044231795255      4.2000000000000011
-9.6677226600969366      4.4000000000000012
-10.5699481642553739      4.6000000000000014
-11.5117702529509884      4.8000000000000016
-12.4932620530009242      5.0000000000000018
Степень интерполяционного многочлена (меньше числа аргументов): 10

#                       f(x)                          x
0.1353352832366133     -1.8646647167633867      1.9999999999999998
0.3091968416376654     -2.3091968416376654      2.1999999999999997
0.5452988882215868     -1.4547011117784132      1.7999999999999998
0.7892820467105874     -2.7892820467105874      2.3999999999999999
0.9218965179946557     -1.0781034820053443      1.5999999999999999
1.2665969639416066     -0.7334030360583934      1.3999999999999999
1.3057264217856663     -3.3057264217856663      2.6000000000000001
1.5811942119122022     -0.4188057880877978      1.2000000000000000
1.8591899373747829     -3.8591899373747829      2.8000000000000003
1.8678794411714423     -0.1321205588285577      1.0000000000000000
2.1293289641172217      0.1293289641172215      0.8000000000000000
2.7987307530779817      0.7987307530779818      0.2000000000000000
2.3688116360940263      0.3688116360940263      0.6000000000000001
2.5903200460356395      0.5903200460356393      0.4000000000000000
3.0000000000000000      1.0000000000000000      0.0000000000000000
2.4502129316321382     -4.4502129316321382      3.0000000000000004
```

```
3.0792377960216353     -5.0792377960216353      3.2000000000000006
3.7466267300396767     -5.7466267300396767      3.4000000000000008
4.4526762775527109     -6.4526762775527109      3.6000000000000010
5.1976292281438390     -7.1976292281438390      3.8000000000000012
5.9816843611112693     -7.9816843611112693      4.0000000000000009
6.8050044231795255     -8.8050044231795255      4.2000000000000011
7.6677226600969366     -9.6677226600969366      4.4000000000000012
8.5699481642553739    -10.5699481642553739      4.6000000000000014
9.5117702529509884    -11.5117702529509884      4.8000000000000016
10.4932620530009242    -12.4932620530009242      5.0000000000000018


Интерполяция Лагранжа:
Значение в точке интерполяции f(x) = -2.0000000000000000: 2.0625835191579713
Абсолютная погрешность: 3.3264725797726791

Интерполяция Ньютона:
Значение в точке интерполяции f(x) = -2.0000000000000000: 2.0625835191579722
Абсолютная погрешность: 3.3264725797726782
```

3.1.3 Второй способ решения

В случае, если выяснить строгую монотонность функции не удаётся, можно воспользоваться знаниями о нахождении корня из ЛР №1. Будем искать корень вида $P_n(x) = F$, где F — точка интерполирования.

В качестве входных параметров:

- функция $f(x) = e^{4.5x}$;
- $a = 0$ - начало отрезка;
- $b = 1$ - конец отрезка;
- $\varepsilon = 10^{-8}$ - погрешность;
- $(m + 1) = 10$ - число аргументов;

3.1.4 Вывод

```
    Второй способ решения
Начало отрезка: 0
Конец отрезка: 1
(Погрешность) eps = 10e-8
(Число разбиений) N = 10

x                       f(x)
0.0000000000000000      1.0000000000000000
0.1111111111111111      0.8886664773081969
0.2222222222222222      0.7760460448921167
0.3333333333333333      0.6609757550182337
0.4444444444444444      0.5424149563311892
0.5555555555555556      0.4194324330831117
0.6666666666666667      0.2911948968103698
0.7777777777777779      0.1569566882334573
0.8888888888888891      0.0160505621121255
1.0000000000000002     -0.1321205588285580

Введите точку интерполяции f(x): -2



Метод секущих:

Между 2.0000000000000013 и 2.1000000000000014:
Начальное приближение: 2.0621089161504704
Число итераций: 3
Приближённое решение: 2.0625787121669803
Длина последнего отрезка: 0.0000000006496084
Невязка |f(X)-0|: 0.0000000008585630

Число корней: 1


Желаете повторить?
Введите цифру 1, чтобы продолжить. Чтобы закончить, введите любое другое число
1
Начало отрезка: 0
Конец отрезка: 1
(Погрешность) eps = 10e-8
(Число разбиений) N = 10

x                       f(x)
```

```
0.0000000000000000      1.0000000000000000
0.1111111111111111      0.8886664773081969
0.2222222222222222      0.7760460448921167
0.3333333333333333      0.6609757550182337
0.4444444444444444      0.5424149563311892
0.5555555555555556      0.4194324330831117
0.6666666666666667      0.2911948968103698
0.7777777777777779      0.1569566882334573
0.8888888888888891      0.0160505621121255
1.0000000000000002     -0.1321205588285580

Введите точку интерполяции f(x): 0.8



Метод секущих:

Между 0.0999999999999999 и 0.1999999999999999:
Начальное приближение: 0.1987446456446144
Число итераций: 2
Приближённое решение: 0.1987539520361326
Длина последнего отрезка: 0.0000000010279224
Невязка |f(X)-0|: 0.0000000000000011

Число корней: 1



Желаете повторить?
Введите цифру 1, чтобы продолжить. Чтобы закончить, введите любое другое число
1
Начало отрезка: 0
Конец отрезка: 1
(Погрешность) eps = 10e-8
(Число разбиений) N = 10

x                       f(x)
0.0000000000000000      1.0000000000000000
0.1111111111111111      0.8886664773081969
0.2222222222222222      0.7760460448921167
0.3333333333333333      0.6609757550182337
0.4444444444444444      0.5424149563311892
0.5555555555555556      0.4194324330831117
0.6666666666666667      0.2911948968103698
0.7777777777777779      0.1569566882334573
0.8888888888888891      0.0160505621121255
```

```
1.0000000000000002     -0.1321205588285580

Введите точку интерполяции f(x): 4



Метод секущих:

Между -1.7000000000000122 и -1.6000000000000121:
Начальное приближение: -1.6919527586392464
Число итераций: 3
Приближённое решение: -1.6924230082819045
Длина последнего отрезка: 0.0000000102439852
Невязка |f(X)-0|: 0.0000000267318683

Число корней: 1
```

3.2 Задача №3.2

В этой задаче, пользуясь формулами численного дифференцирования, определяем значение в точках таблично-заданной функции, а затем выводим модуль разности точной производной от производной, выведенной из формул.

В качестве входных параметров:

- функция $f(x) = e^{4.5x}$;
- $a = 0$ - начало отрезка;
- $h = 0.2$ - шаг отрезка;
- $(m + 1) = 6$ - число аргументов;

3.2.1 Вывод

```
Задание №3.2
Нахождение производных таблично-заданной функции
по формулам численного дифференцирования

Вариант 7: f(x) = exp(4.5*x)

Введите начало отрезка: 0
Введите длину шага h (> 0): 0.2
Введите число аргументов: 6

x                       f(x)
0.0000000000000000      1.0000000000000000
0.2000000000000000      2.4596031111569499
0.4000000000000000      6.0496474644129465
0.6000000000000001     14.8797317248728369
0.8000000000000000     36.5982344436779883
1.0000000000000000     90.0171313005218110

  f'(x)ЧД               |f'(x)ЧД - f'(x)T|
1.9719124505371322      2.5280875494628678
12.6241186610323659      1.5559046608260907
31.0503215342897150      3.8269079444314542
76.3714674481626048      9.4126746862348369
187.8434989391224121     23.1514439425714613
346.3454696293158577     58.7316212230322776

f''(x)ЧД                  |f''(x)ЧД - f''(x)T|
53.2610310524761630                    nan
53.2610310524761630      3.4540680515479281
131.0009976800973277      8.495636525735164
```

```
322.2104614586314142     20.8958940299564802
792.5098534509666024     51.3956059664873237
792.5098534509667161                     nan
```

3.3 Код программы

3.3.1 main.cpp

```
int main(){
    wcout << fixed;
    wcout.precision(16);
    _wsetlocale(LC_ALL, L"russian");

    int num;

    // Task3.1
    introOne();

    wcout << L"Выберите способ решения задачи №3.1\n1. Первый интерполяционный()\n2. Второй через( корни уравнения)\Илиn любое другое число, чтобы выйти" << endl;
    cin >> num;
    switch (num)
    {
    case 1:
        procedureOne();
        break;
    case 2:
        procedureTwo();
        break;
    default:
        break;
    }

    introTwo();

    // Task3.2
    procedureThree();
    return 0;
}
```

3.3.2 Task3.1.cpp

```cpp
#include"Task3.1.h"

////////////////////////////////

void LagrangeInterpolation(double *arg, size_t size, size_t deg, double x, double
    (*fnc)(double*, size_t, double, double), double (*f)(double))
{
    wcout << endl << L"Интерполяция Лагранжа:" << endl;
    double denominator;
    double numerator;
    double total = 0;
    double *ptr = arg;
    for(;ptr < (arg + deg); ptr++)
    {
        denominator = fnc(arg, deg, *ptr, *ptr);
        numerator = fnc(arg, deg, x, *ptr);
        double temp = *(ptr+size)/denominator;
        // cout « "denominator: «< denominator « endl;
        total += numerator*temp;
        // cout « "total: «< total « endl;
        cout << temp;
        for(double *ptrj = arg; ptrj < (arg + deg); ptrj++)
        {
            if(ptrj!=ptr) cout << "(x - " << *ptrj << ")";
        }
        cout << '+';
    }

    /*
    for(size_t i = 0; i < deg; i++)
    {
        double a = 0;
        for(double *ptrj = arg; ptrj <= (arg + i) ; ptrj++)
        {
            denominator = fnc(arg, i, *ptrj, *ptrj);
            a += *(ptrj + size)/denominator;
        }
        cout << a;
        for(size_t j = 0; j < i; j++) cout << "(x - " << arg[j] << ")";
        cout << '+';
    }
    */

    wcout << L"\Значениеn в точке интерполяции f(x) = " << x << ": " << total << endl;
```

```
45      wcout << L"Абсолютная погрешность: " << fabs(total - f(x)) << endl << endl;
46  }
47
48  double LagrangeInterpolationVar(double *arg, size_t size, size_t deg, double x, double
        (*fnc)(double*, size_t, double, double), double (*f)(double))
49  {
50      double denominator;
51      double numerator;
52      double total = 0;
53      double *ptr = arg;
54      for(;ptr < (arg + deg); ptr++)
55      {
56          denominator = fnc(arg, deg, *ptr, *ptr);
57          numerator = fnc(arg, deg, x, *ptr);
58          double temp = *(ptr+size)/denominator;
59          total += numerator*temp;
60      }
61      return total;
62
63  }
64
65  void NewtonInterpolation(double *arg, size_t size, size_t deg, double x, double
        (*fnc)(double*, size_t, double, double), double (*f)(double))
66  {
67      setlocale(LC_ALL, "russian");
68      wcout << L"Интерполяция Ньютона:" << endl;
69      size_t degp = deg + 1;
70      double space[degp][deg], total = 0, product = 1;
71      for(size_t i = 0; i < deg; i++)
72      {
73          space[0][i] = arg[i];
74          space[1][i] = arg[size+i];
75      }
76      for(size_t i = 2; i < degp; i++)
77      {
78          for(size_t j = 0; j < degp - i; j++)
79          {
80              space[i][j] = (space[i-1][j+1]-space[i-1][j])/(space[0][j+i-1]-space[0][j]);
81          }
82      }
83
84      /*
85      for(size_t i = 0; i < 2; i++)
86      {
87          cout << "f" << i << ": ";
88          for(size_t j = 0; j < degp; j++)
89              cout << space[i][j] << ' ';
```

```
90          cout << endl;
91      }
92
93      for(size_t i = 2; i < degp; i++)
94      {
95          cout << "f" << i << ": ";
96          for(size_t j = 0; j < degp - i; j++)
97          cout << space[i][j] << ' ';
98          cout << endl;
99      }
100     */
101
102     for(size_t i = 1; i < degp; i++)
103     {
104         cout << space[i][0];
105         for(int j = i-2; j>=0; j--) cout << "(x - " << space[0][j] << ")";
106         cout << '+';
107     }
108     for(size_t i = 1; i < degp; i++)
109     {
110         total += space[i][0]*product;
111         product *= (x - arg[i-1]);
112     }
113
114     wcout << L"\Значениеn в точке интерполяции fx() = " << x << ": " << total << endl;
115     wcout << L"Абсолютная погрешность: " << fabs(total - f(x)) << endl << endl;
116 }
117
118
119 ////////////////////////////
120
121 double f(double x)
122 {
123     return exp(-x) - x*x/2;
124 };
125
126 double df(double x)
127 {
128     return -exp(-x) - x;
129 }
130
131 double fi(double x, double node, double* arg, size_t N)
132 {
133     return x - fMod(x, node, arg, N)/df(x);
134 }
135
136 double secf(double x, double y, double node, double* arg, size_t N)
```

```
137 {
138     return x - fMod(x, node, arg, N)/(f(x) - f(y))*(x - y);
139 }
140
141 void BiSect(double beg, double end, double h, double eps, double node, double* arg,
        size_t N, double (*f)(double, double, double*, size_t))
142 {
143     size_t counter = 0;
144     double x = beg, xNext = beg + h;
145     while(x < end)
146     {
147         double a = x, b = xNext;
148         double c, delta, fappr=(a+b)/2;
149         size_t m = 0;
150         if(f(a, node, arg, N)*f(b, node, arg, N)<=0)
151         {
152             counter++;
153             while(b - a >= 2*eps){
154                 m++;
155                 c = (a+b)/2;
156                 if(f(b, node, arg, N)*f(c, node, arg, N)<=0)
157                 {
158                     a = c;
159                 } else b = c;
160             }
161             delta = (b-a)/2;
162
163             wcout << '\n' << L"Между " << x << L" и " << xNext << ":\n"<< L"Начальное
        приближение: " << fappr << endl;
164             wcout << L"Число итераций: " << m << L"\Приближённоеn решение: " << c <<
        L"\Длинаn последнего отрезка: " << delta;
165             wcout << L"\Невязкаn |f(X)-0|: " << fabs(f(c, node, arg, N)) << endl;
166
167         }
168
169         x = xNext;
170         xNext += h;
171     }
172     wcout << '\n' << L"Число корней: " << counter << endl;
173 }
174
175 void NewtonApprox(double beg, double end, double h, double eps, double node, double*
        arg, size_t N, double (*f)(double, double, double*, size_t), double (*fi)(double,
        double)){
176     size_t counter = 0;
177     double x = beg, xNext = beg + h;
178     while(x < end)
```

```
179     {
180         double a = x, b = xNext;
181         double c = (a+b)/2, delta, fappr=(a+b)/2;
182         size_t m = 0;
183         if(f(a, node, arg, N)*f(b, node, arg, N) <= 0){
184             counter++;
185             while(fabs(c - a) >= eps){
186                 m++;
187                 a = c;
188                 c = fi(c, node);
189                 if(c > xNext || c < x){
190                     wcout << L"Между "<< x << L" и " << xNext << L": Ошибка: корень не
    может быть вычислен попробуйте( взять N побольше)" << endl;
191                     m = 0;
192                     break;
193                 }
194             }
195         }
196         if(m){
197             delta = fabs(c-a);
198 
199             wcout << '\n' << L"Между " << x << L" и " << xNext << ":\n"<< L"Начальное
    приближение: " << fappr << endl;
200             wcout << L"Число итераций: " << m << L"\Приближённоеn решение: " << c <<
    L"\Длинаn последнего отрезка: " << delta;
201             wcout << L"\Невязкаn |f(X)-0|: " << fabs(f(c, node, arg, N)) << endl;
202 
203         }
204 
205         x = xNext;
206         xNext += h;
207     }
208     wcout << '\n' << L"Число корней: " << counter << endl;
209 }
210 
211 void ModNewtonApprox(double beg, double end, double h, double eps, double node, double*
    arg, size_t N, double (*f)(double, double, double*, size_t), double (*df)(double)){
212     size_t counter = 0;
213     double x = beg, xNext = beg + h;
214     while(x < end)
215     {
216         double a = x, b = xNext;
217         double c = (a+b)/2, delta, fappr=(a+b)/2;
218         double cf = df(c);
219         size_t m = 0;
220         if(f(a, node, arg, N)*f(b, node, arg, N) <= 0){
221             counter++;
```

```
222             while(fabs(c - a) >= eps){
223                 m++;
224                 a = c;
225                 c = c - f(c, node, arg, N)/cf;
226                 if(c > xNext || c < x){
227                     wcout << L"Между "<< x << L" и " << xNext << L": Ошибка: корень не
    может быть вычислен попробуйте( взять N побольше)" << endl;
228                     m = 0;
229                     break;
230                 }
231             }
232         }
233         if(m){
234             delta = fabs(c-a);
235 
236             wcout << '\n' << L"Между " << x << L" и " << xNext << ":\n"<< L"Начальное
    приближение: " << fappr << endl;
237             wcout << L"Число итераций: " << m << L"\Приближённоеn решение: " << c <<
    L"\Длинаn последнего отрезка: " << delta;
238             wcout << L"\Невязкаn |f(X)-0|: " << fabs(f(c, node, arg, N)) << endl;
239 
240         }
241 
242         x = xNext;
243         xNext += h;
244     }
245     wcout << '\n' << L"Число корней: " << counter << endl;
246 }
247 
248 void SecantApprox(double beg, double end, double h, double eps, double node, double*
    arg, size_t N, double (*f)(double, double, double*, size_t), double (*secf)(double,
    double, double, double*, size_t)){
249      size_t counter = 0;
250      double x = beg, xNext = beg + h;
251      while(x < end){
252         double a = x, b = xNext;
253         double c = secf(b, a, node, arg, N), delta, fappr=secf(b, a, node, arg, N);
254         size_t m = 0;
255         if(f(a, node, arg, N)*f(b, node, arg, N)<=0){
256             counter++;
257             while(fabs(c - b) >= eps){
258                 m++;
259                 a = b;
260                 b = c;
261                 c = secf(c, a, node, arg, N);
262                 if(c < x || c > xNext){
```

```
263                     wcout << L"Между "<< x << L" и " << xNext << L": Ошибка: корень не
    может быть вычислен попробуйте( взять N побольше)" << endl;
264                     m = 0;
265                     break;
266                 }
267             }
268         }
269         if(m){
270             delta = fabs(c - b);
271             wcout << '\n' << L"Между " << x << L" и " << xNext << ":\n"<< L"Начальное
    приближение: " << fappr << endl;
272             wcout << L"Число итераций: " << m << L"\Приближённоеn решение: " << c <<
    L"\Длинаn последнего отрезка: " << delta;
273             wcout << L"\Невязкаn |f(X)-0|: " << fabs(f(c, node, arg, N)) << endl;
274 
275         }
276 
277         x = xNext;
278         xNext += h;
279     }
280     wcout << '\n' << L"Число корней: " << counter << endl;
281 }
282 
283 
284 ////////////////////////////////////
285 
286 double productValue(double *arg, size_t size, double x, double c)
287 {
288     double value = 1; double* ptr = arg;
289     for(;ptr < (arg + size); ptr++) {if(c!=*ptr) value *= (x-*ptr);}
290     // cout « "value: «< value « endl;
291     return value;
292 }
293 
294 double* buildup(double(*f)(double), size_t size, double a, double b)
295 {
296     double *temp = new double[2*size];
297     double *ptr = temp;
298     double h;
299     if(size==1) h = (a+b)/2;
300     else h = (b-a)/(size-1);
301     double x = a;
302     wcout << "x" << setw(27) << "f(x)" << endl;
303     for(;ptr < (temp + size); ptr++, x+=h)
304     {
305         *ptr = x;
306         *(ptr + size) = f(x);
```

```
307         wcout << *ptr << setw(24) << *(ptr+size) << endl;
308     }
309     return temp;
310 };
311
312 void sortTable(double *arg, size_t size, size_t deg, double x, void (*fnc1)(double*,
        double*, size_t, size_t), void (*fnc2)(double*, double*, size_t))
313 {
314
315     double temp[size];
316     for(size_t i = 0; i < size; i++) temp[i]=fabs(x-arg[i]);
317     fnc1(temp, arg, size, deg);
318     fnc2(temp, arg, size);
319 };
320
321 void sortArray(double *temp, double *arg, size_t size, size_t deg)
322 {
323     for(size_t i = 0; i < deg; i++)
324     {
325         size_t min = i;
326         for(size_t j = i + 1; j < size; j++)
327         {
328             if(temp[j] < temp[min]) min = j;
329         }
330         if(min!=i)
331         {
332             double temp1 = temp[min], temp2 = arg[min], temp3 = arg[min+size];
333             temp[min] = temp[i];
334             temp[i] = temp1;
335             arg[min] = arg[i];
336             arg[i] = temp2;
337             arg[min+size] = arg[i+size];
338             arg[i+size] = temp3;
339         }
340     }
341 };
342
343 void printArray(double *tmp, double *temp, size_t size)
344 {
345     double *ptr = temp;
346     double *val = tmp;
347     wcout << "#" << setw(27) << "f(x)" << setw(27) << "x" << endl;
348     for(;ptr < (temp + size); ptr++, val++)
349     {
350         wcout << *val << setw(24) << *ptr << setw(24) << *(ptr + size) << endl;
351
352     }
```

```
353 }
354
355 ////////////////////////////////////
356 // Первый способ решения
357
358 void swapColumns(double *arg, size_t size, double x)
359 {
360     wcout << "f(x)" << setw(27) << "x" << endl;
361     double *threshold = arg + size;
362     for(double *ptrfx = arg; ptrfx < threshold; ptrfx++){
363         double x = *(ptrfx + size);
364         *(ptrfx + size) = *ptrfx;
365         *ptrfx = x;
366         wcout << *ptrfx << setw(24) << *(ptrfx + size) << endl;
367     }
368
369 }
370
371 void procedureOne(){
372     wcout << L"Первый способ решения:" << endl;
373     double a, b;
374     size_t m, n;
375     double x;
376     wcout << L"Начало отрезка: ";   cin >> a;
377     wcout << L"Конец отрезка: "; cin >> b;
378     while(b<=a)
379     {
380         wcout << endl << L"Ошибка: конец отрезка меньшеравен- начала. Пожалуйста,
    введите сначала начало отрезка, затем конец." << endl;
381         wcout << endl << L"Начало отрезка: "; cin >> a;
382         wcout << endl << L"Конец отрезка: "; cin >> b;
383     }
384
385     wcout << L"Число аргументов: "; cin >> m;
386
387     // Этап 0.5. Таблица значений
388     // Замечание: так как m не изменяется, можно использовать статический массив.
389     double* arg = buildup(&f, m, a, b);
390
391     wcout << endl << L"Введите точку интерполяции fx(): "; cin >> x; wcout << endl;
392
393     swapColumns(arg, m, x);
394
395     wcout << L"Степень интерполяционного многочлена меньше( числа аргументов): "; cin
    >> n; wcout << endl;
396     while(n>=m)
397     {
```

```
398         wcout << endl << L"Ошибка: такого многочлена не существует. Пожалуйста, введите
    степень меньше числа аргументов: ";
399         wcout << endl; cin >> n; wcout << endl;
400     }
401
402     sortTable(arg, m, n+1, x, &sortArray, &printArray);
403
404     int p;
405     size_t cmp = n + 1, cmpx = x+1;
406     wcout << L"Нажмите 1, чтобы начать процесс интерполяции" << endl << L"Нажмите 0,
    чтобы выйти" << endl;
407     wcout << endl; cin >> p;
408     while(p!=0 && p!=1)
409     {
410             wcout << endl << L"Ошибка: неправильные значения. Пожалуйста, введите 0 или
    1." << endl << endl;
411             cin >> p;
412     }
413
414     while(p)
415     {
416         if(cmp!=n || cmpx!=x)
417         {
418             cmp = n;
419             cmpx = x;
420             LagrangeInterpolation(arg, m, n+1, x, &productValue, &f);
421             NewtonInterpolation(arg, m, n+1, x, &productValue, &f);
422         }
423         else wcout << endl << L"Введена та же степень и точка интерполяционного
    многочлена. Пожалуйста, введите другие значения." << endl;
424
425         wcout << endl << L"Нажмите 1, чтобы начать процесс интерполяции" << endl <<
    L"Нажмите 0, чтобы выйти" << endl;
426         wcout << endl; cin >> p;
427         while(p!=0 && p!=1)
428         {
429             wcout << endl << L"Ошибка: неправильные значения. Пожалуйста, введите 0 или
    1." << endl << endl;
430             cin >> p;
431         }
432         if(p)
433         {
434             wcout << L"Введите точку интерполяции fx(): "; cin >> x; wcout << endl;
435             wcout << L"Степень интерполяционного многочлена меньше( числа аргументов):
    "; cin >> n; wcout << endl;
436             while(n>=m)
437             {
```

```
438                 wcout << endl << L"Ошибка: такого многочлена не существует. Пожалуйста,
    введите степень меньше числа аргументов: ";
439                 wcout << endl; cin >> n; wcout << endl;
440             }
441             sortTable(arg, m, n+1, x, &sortArray, &printArray);
442         }
443     }
444 }
445
446 double fMod(double x,double node, double* arg, size_t N)
447 {
448     double fx = LagrangeInterpolationVar(arg, N, N, x, &productValue, &f);
449     return fx - node;
450 }
451
452 void procedureTwo(){
453     wcout << L"Второй способ решения" << endl;
454     double a, b;
455     double eps;
456     size_t N;
457     double x;
458     int flag = 1;
459     while(!(flag-1))
460     {
461
462         /* 0. Entering data and checking-in */
463         wcout << L"Начало отрезка: ";   cin >> a;
464         wcout << L"Конец отрезка: "; cin >> b;
465         while(b<=a)
466         {
467             wcout << endl << L"Ошибка: конец отрезка меньшеравен- начала. Пожалуйста,
    введите сначала начало отрезка, затем конец." << endl;
468             wcout << endl << L"Начало отрезка: "; cin >> a;
469             wcout << endl << L"Конец отрезка: "; cin >> b;
470         }
471         wcout << L"Погрешность() eps = ";cin >> eps;
472         while(eps <= 0)
473         {
474             wcout << endl << L"Ошибка: отрицательное значение погрешности" << endl;
475         }
476         wcout << L"Число( разбиений) N = "; cin >> N; wcout << endl;
477         while(N == 0)
478         {
479             wcout << endl << L"Ошибка: неположительное число узлов интерполирования.
    Пожалуйста, введите N не менее 1." << endl;
480             cin >> N;
481         }
```

```
482
483         double* arg = buildup(&f, N, a, b);
484
485         /* 1. Decoupling roots on [A, B] */
486
487         double h = (b-a)/N;
488         wcout << endl << L"Введите точку интерполяции fx(): "; cin >> x; wcout << endl;
489         // 1.1 Bisection Method
490         /**
491         wcout << '\n' << '\n' << LМетод" бисекции: " << endl;
492         BiSect(a, b, h, eps, x, fMod);
493         */
494
495         /**
496         // 1.2 Tangent Method
497         wcout << '\n' << '\n' << LМетод" Ньютона: " << endl;
498         NewtonApprox(a, b, h, eps, x, fMod, fi);
499         */
500
501         /**
502         // 1.3 Modified Tangent Method (MoTangeM)
503         wcout << '\n' << '\n' << LМодифицированный" метод Ньютона: " << endl;
504         ModNewtonApprox(a, b, h, eps, x, fMod, df);
505         */
506
507         // 1.4 Secant Method
508         wcout << '\n' << '\n' << L"Метод секущих: " << endl;
509         SecantApprox(a, b, h, eps, x, arg, N, fMod, secf);
510
511         wcout << endl << endl << L"Желаете повторить?\Введитеn цифру 1, чтобы
    продолжить. Чтобы закончить, введите любое другое число" << endl;
512         cin >> flag;
513     }
514 }
515
516 void introOne(){
517     wcout << L"Задача обратного интерполирования:" << endl;
518     wcout << L"Интерполяционный многочлен в форме Ньютона и в форме Лагранжа." << endl
    << endl;
519     wcout << L"Вариант 7. Функция f(x) = eps(-x) - x^2/2" << endl << endl;
520 }
```

3.3.3 Task3.1.h

```
#ifndef TASK3_1_H_INCLUDED
#define TASK3_1_H_INCLUDED

#include<locale.h>
#include<iostream>
#include<math.h>
#include<iomanip>
using namespace std;


void LagrangeInterpolation(double *arg, size_t size, size_t deg, double x, double
    (*fnc)(double*, size_t, double, double), double (*f)(double));

double LagrangeInterpolationVar(double *arg, size_t size, size_t deg, double x, double
    (*fnc)(double*, size_t, double, double), double (*f)(double));

void NewtonInterpolation(double *arg, size_t size, size_t deg, double x, double
    (*fnc)(double*, size_t, double, double), double (*f)(double));

double productValue(double *arg, size_t size, double x, double c);

double* buildup(double(*f)(double), size_t size, double a, double b);

void sortTable(double *arg, size_t size, size_t deg, double x, void (*fnc1)(double*,
    double*, size_t, size_t), void (*fnc2)(double*, double*, size_t));

void sortArray(double *temp, double *arg, size_t size, size_t deg);

void printArray(double *tmp, double *temp, size_t size);

double f(double x);

double df(double x);

double ddf(double x);

double fi(double x);

double dfi(double x);

double secf(double x, double y, double node, double* arg, size_t N);

void BiSect(double beg, double end, double h, double eps, double node, double* arg,
    size_t N, double (*f)(double, double, double*, size_t));

```

```
41 void NewtonApprox(double beg, double end, double h, double eps, double node, double*
       arg, size_t N, double (*f)(double, double), double(*fi)(double, double));
42
43 void ModNewtonApprox(double beg, double end, double h, double eps, double node, double*
       arg, size_t N, double (*f)(double, double, double*, size_t), double(*df)(double));
44
45 void SecantApprox(double beg, double end, double h, double eps, double node, double*
       arg, size_t N, double (*f)(double, double, double*, size_t), double(*secf)(double,
       double, double, double*, size_t));
46
47 void introOne();
48
49 void swapColumns(double *arg, size_t size, double x);
50
51 void procedureOne();
52
53 void procedureTwo();
54
55 double fMod(double x, double node, double* arg, size_t N);
56
57 #endif // TASK3_1_H_INCLUDED
```

3.3.4 Task3.2.cpp

```cpp
#include"Task3.2.h"

double f2(double x)
{
    return exp(4.5*x);
}

double* buildup(double(*f)(double), size_t size, double a, double *h)
{
    double *temp = new double[2*size];
    double *ptr = temp;
    wcout << "x" << setw(27) << "f(x)" << endl;
    for(;ptr < (temp + size); ptr++, a+=*h)
    {
        *ptr = a;
        *(ptr + size) = f(a);
        wcout << *ptr << setw(24) << *(ptr+size) << endl;
    }
    return temp;
};

double* firstDer(double(*f)(double), double const* arg, size_t size, double h)
{
    double *temp = new double[2*size];
    double *ptr = temp;
    size_t t = 0;
    // 1 2 3 4
    // 4 3 2 1
    // 1 2 3 4 4 3 2 1
    *ptr = (-3*arg[size] + 4*arg[size + 1] - arg[size + 2])/(2*h);
    *(ptr + size) = fabs(4.5*f(arg[0]) - *ptr);
    ptr++; t++;
    for(; ptr < (temp + size - 1); ptr++, t++){
        *ptr = (arg[size + t + 1] - arg[size + t - 1])/(2*h);
        *(ptr + size) = fabs(4.5*f(arg[t]) - *ptr);
    }
    *ptr  = (3*arg[size + t] - 4*arg[t + size - 1] + arg[t + size - 2])/(2*h);
    *(ptr + size) = fabs(4.5*f(arg[t]) - *ptr);
    return temp;
}

double* secondDer(double(*f)(double), double const* arg, size_t size, double h)
{
    double *temp = new double[2*size];
    double *ptr = temp;
```

```
46      double const *ptrTemp = arg;
47      *ptr = (*(ptrTemp + size) - 2**(ptrTemp + size + 1) + *(ptrTemp + size + 2))/(h*h);
48      *(ptr + size) = 0/0.;
49      ptr++; ptrTemp++;
50      for(; ptr < (temp + size - 1); ptr++, ptrTemp++){
51          *ptr = (*(ptrTemp + size + 1) - 2**(ptrTemp + size) + *(ptrTemp + size -
        1))/(h*h);
52          *(ptr + size) = fabs(20.25*f(*ptrTemp) - *ptr);
53      }
54      *ptr  = (*(ptrTemp + size - 2) - 2**(ptrTemp + size - 1) + *(ptrTemp + size))/(h*h);
55      *(ptr + size) = 0/0.;
56      return temp;
57  }
58
59  void printTable(double *arg, double *fD, double *sD, size_t size)
60  {
61      wcout << "x" << setw(27) << "f(x)" << setw(27) << L"f'(x)ЧД" << setw(27);
62      wcout << L"|f'(x)ЧД - f'(x)Т|" << setw(27) << L"f''(x)ЧД" << setw(27) <<
        L"|f''(x)ЧД - f''(x)Т|" << endl;
63      double *tempA = arg;
64      double *tempB = fD;
65      double *tempC = sD;
66      for(; tempA < (arg + size); tempA++, tempB++, tempC++)
67      {
68          wcout << *tempA << setw(24) << *(tempA + size) << setw(24) << *tempB <<
        setw(24) << *(tempB + size);
69          wcout << setw(24) << *tempC << setw(24) << *(tempC + size) << endl;
70      }
71  }
72
73
74  void introTwo(){
75      wcout << L"Задание №3.2\Нахождениеn производных табличнозаданной- функции\n";
76      wcout << L"по формулам численного дифференцирования" << endl << endl;
77      wcout << L"Вариант 7: f(x) = exp(4.5*x)" << endl << endl;
78  }
79
80  void procedureThree(){
81
82      size_t m;
83      double a, h;
84      int flag = 1;
85      while(!(flag - 1))
86      {
87          wcout << L"Введите начало отрезка: "; cin >> a;
88          wcout << L"Введите длину шага h (> 0): "; cin >> h;
89
```

```
90          while(h<=0)
91          {
92              wcout << L"Ошибка: отрицательный шаг. Пожалуйста, введите положительное
    значение шага.";
93              wcout << endl << "h = "; cin >> h;
94          }
95          wcout << L"Введите число аргументов: "; cin >> m;
96
97          while(m <= 3)
98          {
99              wcout << L"Ошибка: недостаточное число аргументов. Пожалуйста, введите
    положительное значение более трёх.";
100             wcout << endl << "(m + 1) = "; cin >> m;
101         }
102
103         double *arg = buildup(f2, m, a, &h);
104         double *firstDerivative = firstDer(f2, arg, m, h);
105         double *secondDerivative = secondDer(f2, arg, m, h);
106         printTable(arg, firstDerivative, secondDerivative, m);
107         wcout << endl << endl;
108         /*
109         for(size_t tt = 0; tt < m; tt++)
110         {
111             wcout << 4.5*f2(arg[tt]) << setw(24) << firstDerivative[tt] << endl;
112         }
113         */
114         wcout << L"Хотите начать снова? Введите 1, чтобы продолжить и 0, чтобы
    прекратить." << endl;
115         cin >> flag;
116     }
117
118 }
```

3.3.5 Task3.2.h

```
#ifndef TASK3_2_H_INCLUDED
#define TASK3_2_H_INCLUDED

#include<locale.h>
#include<iostream>
#include<math.h>
#include<iomanip>
using namespace std;

double f2(double x);

double* buildup(double(*f)(double), size_t size, double a, double *h);

double* firstDer(double(*f)(double), double const* arg, size_t size, double h);

double* secondDer(double(*f)(double), double const* arg, size_t size, double h);

void introTwo();

void procedureThree();
#endif // TASK3_2_H_INCLUDED
```