Оглавление

[**Задание** 4](#_Toc134894967)

[**Алгоритм решения** 5](#_Toc134894968)

[**Теоретические сведения** 6](#_Toc134894969)

[**Вычисление функции** 7](#_Toc134894970)

[**Поиск минимума функции** 8](#_Toc134894971)

[**Приложение** 11](#_Toc134894972)

# **Задание**

Вариант 6

Вычислить минимум функции на отрезке [a, b] с точностью ε P(x) - интерполяционный многочлен для функции f(x), заданной таблично; k = P(c).

Исходные данные:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1.05 | 1.15 | 1.25 | 1.35 |
|  | 2.30 | 2.74 | 3.46 | 4.60 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 1.05 | 1.35 | 1.10 | 0.001 |

# **Алгоритм решения**

Сначала необходимо вычислить функцию F(x). Для этого посчитаем интерполяционный многочлен Лагранжа. Затем вычислить значение полученного многочлена в точке c. Далее требуется посчитать произведение этого значение, многочлена Лагранжа и данного многочлена 1.05x + 1.35 – это и будет функцией F(x).

Затем будет посчитано минимальное значение функции на отрезке [1.05, 1.35]. Это будет сделано 2 способами: аналитически и перебором.

# **Теоретические сведения**

Интерполяционный многочлен Лагранжа

Интерполяционный многочлен Лагранжа — многочлен минимальной степени,

принимающий данные значения в данном наборе точек. Для n+1 пар чисел (x0, y0),(x1, y1),…, (xn, yn), где все xj различны, существует единственный многочлен L(x)степени не более n, для которого L(xj) = yj.

Многочлен Лагранжа вычисляется следующим образом:

, где

=

Многочлен удовлетворяет условию = . Это условие означает, что многочлен равен нулю при каждом кроме http://aco.ifmo.ru/el_books/numerical_methods/lectures/images/image148.png – корни этого многочлена. Таким образом, степень многочлена равна n и при x обращаются в ноль все слагаемые суммы, кроме слагаемого с номером равного .

Существует единственный многочлен степени не превосходящей n, принимающий

заданные значения в n+1 точке (что является обобщением факта, что через любые две

точки проходит единственная прямая). Действительно, предположим, что существуют

два различных многочлена степени не более n: P(x) и Q(x), для которых верно, что для

n+1 пар чисел (, где все различны (P()=Q()=). Рассмотрим многочлен L (x) = P( x )-Q( x ). Подставляя вместо x получаем, что L() = P()-Q()= Таким образом, многочлен L(x) имеет n+1 корней, и все они различны. Следовательно, L(x)=0, так как ненулевой многочлен степени, не превосходящей n имеет не более n корней. Следовательно, P(x) = Q(x).

# **Вычисление функции**

Во-первых, необходимо вычислить интерполяционный многочлен. Он был вычислен с помощью нескольких программ, написанных на языке C++.

Основная программ ***inter\_mg*** ( её код, как и код всех последующих программ представлен в Приложении) принимает на вход два вектора: вектор x ( вектор точек, в которых вычисляется значение функции) и вектор y ( вектор со значениями функции в точках). На выходе мы получаем вектор со значениями коэффициентов многочлена (нулевой элемент – число при максимальной степени x и т.д. ( в данном случае, мы получаем многочлен 3 степени); последний элемент – это свободный член).

Для вычисления интерполяционного многочлена требовалось вычисление всевозможных сочетаний. Для этого была написана программа ***number\_generator*** . На вход она принимает два числа n и k. n – это количество чисел, из которых составляются сочетания. При этом само число n в данное множество не входит (если n будет равно 5, то сочетания будут составляться из чисел 0, 1, 2, 3, 4). Это связано с тем, что сочетания будут использоваться как номера элементов массива, поэтому используется 0. k – это длина сочетания. На выход подаётся вектор векторов (например, если n = 3 и k = 3, то в ответ мы получим всего один вектор 012, если же n = 3, а k = 1 то в ответ мы получим 3 вектора: 0, 1, 2.)

Теперь взглянем на результаты работы программы ***inter\_mg.***  Она принимает два массива: x = [1.05, 1.15, 1.25, 1.35] и y = [2.30, 2.74, 3.46, 4.60]. На выходе мы получаем массив со следующими коэффициентами: [23.3333, -66.5, 65.9417, -20.6338].

Получается, интерполяционный многочлен Лагранжа равен:

**-66.5 + 65.9417x -20.6338.**

Во-вторых, требуется вычислить значение данного многочлена в точке c = 1.10. Для этого была написана программа ***ans\_in\_dote.*** На вход она принимает вектор коэффициентов многочлена и точку, в которой нужно вычислить значение. На выходе мы получаем одно число.

В результате работы программы получаем, что значение многочлена P(x) в точке c = 1.10 равно:

**2.49375.**

И последний шаг нужно перемножить значение многочлена в точке c (2.49375), интерполяционный многочлен Лагранжа -66.5 + 65.9417x -20.6338) и функции ax+b (1.05x+1.35). Чтобы упростить задачу умножим функцию ax+b на k, тогда основная задача состоит в перемножении двух функций. Для этого была написана программа mg\_multiplier. На вход она принимает вектор коэффициентов многочлена и два числа (в данном случае это ak и bk). На выходе мы получаем новый многочлен.

В итоге, функция для которой необходимо вычислить минимум на отрезке [a, b] равна:

# **Поиск минимума функции**

Поиск минимума будет осуществлен 2 способами.

Первый способ – аналитический. Сначала будет высчитана производная нашей функции. Сделано это будет с помощью программы proiz. В результате производная будет равняться:

= **-286.719 -102.425x + 167.969.**

Далее нам нужно найти точки экстремума. Для этого необходимо найти, в каких точках равняется 0. (рассматриваться будут только действительные точки). Будем использовать программу ***eqn\_cubic.*** Она находит всё действительные корни с помощью метода Галлея.

У производной от нашего многочлена только один корень действителен. Он равен:

Теперь нужно вычислить значение изначального многочлена во всех точках экстремума (которые лежат на отрезке [a, b] ) и на краях отрезка (точках a и b). Поскольку точка на отрезке не лежит значение в ней вычисляется не будет. Проделано это будет с помощью функции min\_metod1. Она принимает коэффициенты нашего многочлена, точки экстремума и точки a и b.

В результате, получаем, что минимум функции на отрезке [a, b] равен **14.067**.

Теперь найдем минимум вторым способом. Он гораздо проще первого, но при этом гораздо более временно затратный. Для нахождения минимума мы просто переберем всё точки от a до b с шагом в ε = 0.001 (что является нашей точностью). Стоит отметить, что использование данного метода возможно благодаря небольшой длине отрезка и маленькой точности. В другой задаче вычисление минимума данным способом могло занять слишком много времени. Но в нашей задаче использование этого метода допустимо, и с его помощью вполне можно провести проверку. Вычисление будет проводится с помощью функции min\_metod2.

В результате, получаем, что минимум функции на отрезке [a, b] вновь равен **14.067**.

**Заключение**

Передо мной была поставлена задача вычислить минимум функции на отрезке [1.05, 1.35].

Для решения данной задачи была найдена сама функция равная :

Минимум данной функции был найдем двумя методами: аналитически и переборам – и в обоих методах он равнялся **14.067**.

Для решения данной задачи было написаны программы на языке C++. С их помощью можно: составить многочлен Лагранжа, найти корни кубического уравнения, получить минимум функции 2 способами.

**Список использованной литературы**

1. Вержбицкий В.М. Основы численных методов: Учебник для вузов/В.М. Вержбицкий.-2-е изд., перераб.-М.:Высш. шк., 2002.-840с.
2. Киреев В.И. Численные методы в примерах и задачах: Учеб. Пособие/В.И. Киреев, А.В. Пантелеев.-2-е изд. Стер.- М.:Высш. шк., 2006.-480с..:ил.
3. Волков Е.А Численные методы: Учебное пособие.-3-е изд., испр.-СПБ.:Лань, 2004.-256 с.

# **Приложение**

#include <iostream>

#include <vector>

#include <cmath>

#include <random>

#include <chrono>

using namespace std;

vector<vector<int>> number\_generator(int n, int k)

{

vector <vector<int>> answer;

vector<int> temp;

if (k == 1)

{

for (int i = 0; i < n; i++)

{

temp.push\_back(i);

answer.push\_back(temp);

temp.erase(temp.begin());

}

}

else

{

int z = 0;

for (int i = 0; i < n; i++)

{

temp.push\_back(i);

for (int j = i; j < n; j++)

{

z = j + 1;

while (size(temp) < k && z < n)

{

temp.push\_back(z);

z++;

}

if (size(temp) == k)

{

answer.push\_back(temp);

}

temp.erase(temp.begin() + 1, temp.end());

}

temp.erase(temp.begin());

}

}

return(answer);

}

double multiplier(vector<double> x, vector<vector<int>> i)

{

double answer = 0;

double temp = 1;

for (int j = 0; j < size(i); j++)

{

for (int k = 0; k < size(i[j]); k++)

{

temp = temp \* -x[i[j][k]];

}

answer += temp;

temp = 1;

}

return answer;

}

vector<double> inter\_mg(vector<double> x, vector<double> y)

{

vector<double> answer(size(x));

vector<double> tx(size(x));

vector<double> temp\_answer(size(x));

vector<vector<int>> index;

double tempy = 1;

for (int i = 0; i < size(x); i++)

{

tx = x;

tx.erase(tx.begin() + i, tx.begin() + i+1);

tempy = 1;

for (int j = 0; j < size(tx); j++)

{

tempy = tempy\*(x[i] - tx[j]);

}

for (int j = size(tx); j > 0; j--)

{

index = number\_generator(size(tx), j);

temp\_answer[j] = multiplier(tx, index);

}

temp\_answer[0] = 1;

tempy = y[i] / tempy;

for (int j = 0; j < size(x); j++)

{

temp\_answer[j] = temp\_answer[j] \* tempy;

answer[j] += temp\_answer[j];

}

}

return answer;

}

double ans\_in\_dote(vector<double> x, double c)

{

double temp = 0;

for (int i = 0; i < size(x); i++)

{

temp += x[i] \* pow(c, size(x) - i - 1);

}

return temp;

}

vector<double> mg\_multiplier(vector<double> coef, double a, double b)

{

vector<double> answer(size(coef) + 1);

answer[0] = coef[0] \* a;

for (int i = 1; i < size(coef); i++)

{

answer[i] += coef[i] \* a + b \* coef[i - 1];

}

answer[size(coef)] = coef[size(coef) - 1] \* b;

return answer;

}

double min\_metod2(vector<double> coef, double a, double b, double error)

{

double answer = std::numeric\_limits<double>::infinity();

while (a <= b)

{

answer = min(answer, ans\_in\_dote(coef, a));

a += error;

}

return answer;

}

double min\_metod1(vector<double> coef, vector<double> zero, double a, double b)

{

double answer = std::numeric\_limits<double>::infinity();

for (int i = 0; i < size(zero); i++)

{

if (zero[i] >= a && zero[i] <= b)

{

answer = min(answer, ans\_in\_dote(coef, zero[i]));

}

}

answer = min(answer, ans\_in\_dote(coef, a));

answer = min(answer, ans\_in\_dote(coef, b));

return answer;

}

vector<double> proiz(vector<double> coef)

{

vector<double> answer(size(coef) -1);

for (int i = 0; i < size(coef)-1; i++)

{

answer[i] += coef[i] \* (size(coef)-1-i);

}

return answer;

}

inline double sq(const double& x) { //Функция sq принимает double число, служит для быстрого вычисления квадрата числа

return x \* x;

}

inline int signum(const double& x) {//Функция signum, принимает double число, и возвращает его знак, то есть 1 если число положительно или -1 - если отрицательное

return (0.0 < x) - (x < 0.0);

}

inline double newton1(const double a, const double b, const double c, double x) {//Функция newton1 вычисляет первый шаг итерации Ньютона, принимает два double числа

double y, y1;

y = a + x;

y1 = 2 \* y + x;

y1 = x \* y1 + b;

y = (x \* y + b) \* x + c;

if (y1 != 0.0) x -= y / y1;

return x;

}

inline int eqn\_quadratic(const double a, const double b, double\*& x) {//Функция для нахождения корней квадратного урванения, принимает вектор x, в него заносятся ответы, и 2 числа double, a это коэфициент при x, а b это свободный член

double p = -0.5f \* a,

d = sq(p) - b;//Вычисление дискриминанта

if (d >= 0.0) {//Вычисление вещественных корней

d = sqrt(d);

x[1] = p - d;

x[2] = p + d;

return 2;

}

x[1] = x[2] = std::numeric\_limits<double>::infinity();

return 0;//Если корни комплексные возвращаем 0

}

inline int eqn\_quadratic(const double a, const double b, const double c, const double e, const double f, double\*& x) {//Функция для нахождения корней квадратного уравнения, принимает вектор x, в него заносятся ответы

//2 числа double, a это коэфициент при x, а b это свободный член

//3 числа double,с , e, f, это коэфициенты многочлена третей степени, идущие по убыванию, они необходимы для функции newton1

double p = -0.5f \* a,

d = sq(p) - b;

if (d >= 0.0) {//Вычисляем дискриминат

d = sqrt(d);

if (p < 0.0) {

x[1] = newton1(c, e, f, p - d);//Вычисляем один из корней, с помощью метода Ньютона

x[2] = p + d;

return 2;

}

else {

x[1] = p - d;

x[2] = newton1(c, e, f, p + d);//Вычисляем один из корней, с помощью метода Ньютона

return 2;

}

}

x[1] = x[2] = std::numeric\_limits<double>::infinity();

return 0;//Если корни комплексные возвращаем 0

}

int eqn\_cubic(const double a, const double b, const double e, double\*& x) { // Remark #2

int i\_slope, i\_loc;

double w, xh, y, y1, y2, dx, c[2], d;

double prec = 1.0e-4f; // termination criterion, Remark #3

w = (double)1.0f;

if (e == 0.0) { // Отсутствует ли свободный член

x[0] = 0.0;

if (eqn\_quadratic(a, b, x) == 0)

return 1;

else {

return 3;

}

}

xh = -1.0f / 3.0f \* a; // точка перегиба

y = e + xh \* (b + xh \* (a + xh));

if (y == 0.0) { // Является ли точка перегиба корнем

x[0] = x[1] = xh;

c[1] = xh + a; // Понижение порядка уравнения

c[0] = c[1] \* xh + b;

return 1 + eqn\_quadratic(c[1], c[0], x);

}

i\_loc = (y >= 0.0f);

d = sq(a) - 3 \* b;

if ((i\_slope = signum(d)) == 1) // Laguerre-Nair-Samuelson bounds

{

if (d < 0)

return 0;

else

xh += ((i\_loc) ? -2.0f / 3.0f : 2.0f / 3.0f) \* sqrt(d);

}

else if (i\_slope == 0) { // Седловая точка?

x[0] = xh - cbrt(y);//Вещественный корень всего один

return 1;

}

do { // Итерации ( Сам метод Галлея)

y = a + xh;

y1 = 2 \* y + xh;

y2 = y1 + 3 \* xh;

y1 = xh \* y1 + b;

y = (xh \* y + b) \* xh + e;

dx = y \* y1 / (sq(y1) - 0.5f \* y \* y2);

if (isinf(dx) || isnan(dx))

return 0;

xh -= dx;

} while (fabs(dx) > prec \* fabs(xh));

x[0] = x[2] = xh;

if (i\_slope == 1) {

c[1] = xh + a; // Понижение порядка уравнения

c[0] = c[1] \* xh + b;

return 1 + eqn\_quadratic(c[1], c[0], a, b, e, x);

}

return 1;

}

int main()

{

vector<double> x = { 1.05, 1.15, 1.25, 1.35 };

vector<double> y = { 2.30, 2.74, 3.46, 4.60 };

vector<double> coef = inter\_mg(x, y);

/\*for (int i = 0; i < size(coef); i++)

{

cout << coef[i] << "\n";

}

cout << ans\_in\_dote(coef, 1.05) << "\n";

cout << ans\_in\_dote(coef, 1.15) << "\n";

cout << ans\_in\_dote(coef, 1.25) << "\n";

cout << ans\_in\_dote(coef, 1.35) << "\n";\*/

double a = 1.05;

double b = 1.35;

double c = 1.10;

double error = 0.001;

double k = ans\_in\_dote(coef, c);

/\*cout << k << "\n";\*/

coef = mg\_multiplier(coef, a \* k, b \* k);

double min2 = min\_metod2(coef, a, b, error);

/\*for (int i = 0; i < size(coef); i++)

{

cout << coef[i] << "\n";

}\*/

vector<double> proizv = proiz(coef);

for (int i = 0; i < size(proizv); i++)

{

cout << proizv[i] << "\n";

}

double\* temp = new double [3];

vector<double> dot\_0(3);

eqn\_cubic(proizv[1] / proizv[0], proizv[2] / proizv[0], proizv[3] / proizv[0], temp);

for (int i = 0; i < 3; i++)

{

dot\_0[i] =temp[i];

cout << temp[i] << "\n";

}

double min = min\_metod1(coef, dot\_0, a, b);

cout << min << "\n";

cout << min2 << "\n";

}