МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВПО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Методы приближения функций

Отчет по лабораторной работе №3 дисциплины

«Вычислительная математика»

Выполнил студент группы ИВТб-21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Седов М.Д./

Проверил преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Исупов К.С./

Киров 2018

1 Постановка задачи

По таблице с неравноотстоящими значениями аргумента выполнить интерполяцию, используя формулу Лагранжа. Точность E<=0,000001.

Задание:

X=0,453

0,35 2,73951

0,41 2,30080

0,47 1,96864

0,51 1,78776

0,56 1,59502

0,64 1,34310

По таблице с равностоящими значениями аргумента вычислить значения функции для заданных значений аргументов, используя первую и вторую интерполяционные формулы Ньютона. Точность E<=0.000001.

Задание:

X1=0,1535; X2=0,6247; X3=0,1317; X4=0,6672;

0,15 0,860708

0,20 0,818731

0,25 0,778801

0,30 0,740818

0,35 0,704688

0,40 0,670320

0,45 0,637628

0,50 0,606531

0,55 0,576950

0,60 0,548812

0,65 0,522046

По заданным экспериментальным точкам выбрать вид эмпирической зависимости и выполнить среднеквадратичное приближение функции, применив метод наименьших квадратов для оценки параметров выбранной зависимости.

Задание:

10 95

11 116

12 139

13 163

14 190

15 219

16 250

17 283

18 319

19 355

Проверить результаты с помощью системы Mathcad.

2 Краткие теоретические сведения

2.1 Формула 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 xi различны, существует единственный многочлен L(x) степени не более n, для которого L(xj)=yj.

В простейшем случае (n=1) – это линейный многочлен, график которого – прямая, проходящая через две заданных точки.

Формула Лагранжа имеет следующий вид:

*,* где

базисные полиномы определяются по формуле:

li(x) обладают следующими свойствами:

* являются многочленами степени n
* li(xi) = 1
* li(xj) = 0 при j ≠ i

Отсюда следует, что L(x), как линейная комбинация li(x) может иметь степень не больше n, и L(xi) = yi.

Пусть R(x) = f(x) – L(x) – остаточный член интерполяционного многочлена Лагранжа, и f(n)(x) непрерывна. Тогда:

2.2 Формула Ньютона

Интерполяционные формулы Ньютона применяются в вычислительной математике для полиномиального интерполирования. Если узлы интерполяции равностоящие и упорядочены по величине, так что xi+1 - xi = const, то есть xi = x0 + *ih,* то интерполяционный многочлен можно записать в форме Ньютона.

Интерполяционные полиномы в форме Ньютона удобно использовать, если точка интерполирования находится вблизи начала (прямая формула Ньютона) или конца таблицы (обратная формула Ньютона).

Прямая (или первая) интерполяционная формула Ньютона, применяется при интерполировании вперед:

Обратная (или вторая) интерполяционная формула Ньютона применяется для интерполирования назад:

Величину Rn(x) = |f(x) – Pn(x)| называют погрешностью интерполяции или остаточным членом интерполяции. Если функция дифференцируема n+1 раз на отрезке [a,b], содержащем узлы интерполяции xi, i=0,1,…,n, то для погрешности интерполяции справедлива оценка:

Эта оценка показывает, что для достаточно гладкой функции при фиксированной степени интерполяционного многочлена погрешность интерполяции стремится к нулю не медленнее, чем величина, пропорциональная Этот факт формулируют так: интерполяционный многочлен степени n аппроксимирует функцию с (n+1) порядком точности относительно max.

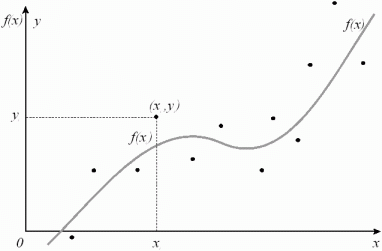
2.3 Средне квадратичное приближение функции

Пусть значения приближаемой функции f(x) заданы в N+1 узлах f(x0),…, f(xn). Аппроксимирующую функцию будем выбирать из некоторого параметрического семейства F(x, c), где c = (c0, …, cn)T – вектор параметров, N > n.

Принципиальным отличием задачи среднеквадратичного приближения от задачи интерполяции является то, что число узлов превышает число параметров. В данном случае практически всегда не найдется такого вектора параметров, для которого значения аппроксимирующей функции совпадали бы со значениями аппроксимирующей функции во всех узлах.

В этом случае задача аппроксимации ставится как задача поиска такого вектора параметров c = (c0, …, cn)T, при котором значения аппроксимирующей функции как можно меньше отклонялись бы от значений аппроксимирующей функции F(x, c) в совокупности всех узлов.

Графически задачу можно представить так:



Запишем критерий среднеквадратичного приближения для метода наименьших квадратов:

Подкоренное выражение представляет собой квадратичную функцию относительно коэффициентов аппроксимирующего многочлена. Она непрерывна и дифференцируема по c0, …, cn. Очевидно, что ее минимум находится в точке, где все частные производные равны нулю. Приравнивая к нулю частные производные, получим систему линейных алгебраических уравнений относительно неизвестных (искомых) коэффициентов многочлена наилучшего приближения.

Метод подразделяется на два основных этапа:

1. Определения типа неизвестно функции, при которой будет наилучшее приближение.
2. Определения неизвестных коэффициентов выбранного многочлена путём решения системы линейных алгебраических уравнений.
3. Для определения типа воспользуемся приближенным методом. Для начала вычислим средние арифметические, геометрические и гармонические по x и по y для начального и конечного заданных значений.

После чего вычисляются значения аппроксимирующей функции для трёх x, посчитанных выше.

Для определения типа зависимости вычисляются 7 погрешностей:

Вывод:

* Если ε= ε1 – линейная зависимость -> y=ax+b
* Если ε= ε2 – показательная зависимость -> y=abx
* Если ε= ε3 – дробно-линейная зависимость -> y=1/(ax+b)
* Если ε= ε4 –логарифмическая зависимость -> y=alnx+b
* Если ε= ε5 -> y=axb
* Если ε= ε6 -> y=a+b/x
* Если ε= ε7 -> y=x/(ax+b)

1. Наименьшие коэффициенты для наилучшей аппроксимации определяются при помощи метода наименьших квадратов

В качестве ответа выписывается аппроксимирующий многочлен с вычисленными коэффициентами во втором этапе.

3 Выполнение задания

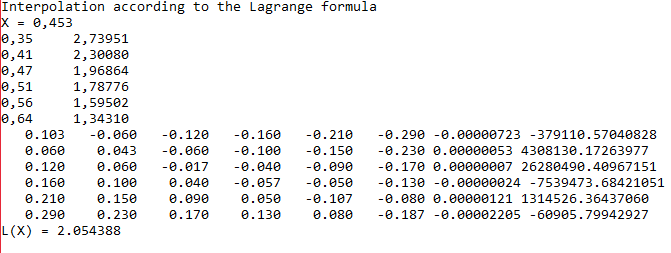


Рисунок 1 – интерполяционный многочлен Лагранжа

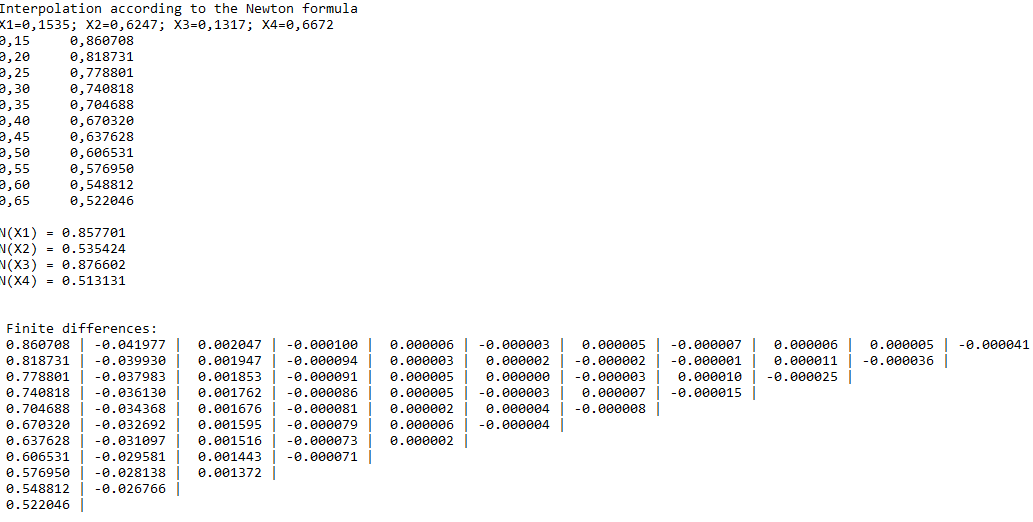


Рисунок 2 – интерполяционный многочлен Ньютона

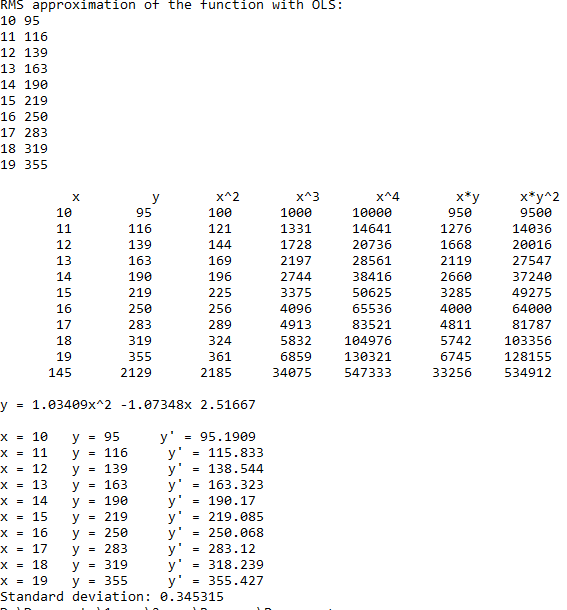


Рисунок 3 –среднеквадратическое приближение

Приложение А

Листинг программы #1

#include <iostream>

#include <iomanip>

#include <cmath>

#include <algorithm>

using namespace std;

const int n = 6;

double lagrang(double \*x, double \*y, double X);

int main() {

double X[6] = { 0.35, 0.41, 0.47, 0.51, 0.56, 0.64 };

double Y[6] = { 2.73951, 2.30080, 1.96864, 1.78776, 1.59502, 1.34310 };

double t = 0.453;

double e = 0.000001;

cout << "Interpolation according to the Lagrange formula" << endl;

cout << "X = 0,453" << endl;

cout << "0,35 2,73951 " << endl

<< "0,41 2,30080 " << endl

<< "0,47 1,96864 " << endl

<< "0,51 1,78776 " << endl

<< "0,56 1,59502 " << endl

<< "0,64 1,34310 " << endl;

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

double p = 1;

for (int j = 0; j < n; j++) {

if (i != j) {

p\*= X[i] - X[j];

cout << fixed << setprecision(3) << setw(8) << X[i] - X[j] << " ";

} else {

p\*= t - X[i];

cout << fixed << setprecision(3) << setw(8) << t - X[i] << " ";

}

}

cout << fixed << setprecision(8) << setw(10) << p << " " << Y[i] / p;

cout << endl;

}

cout << "L(X) = " << fixed << setprecision(6) << lagrang(X, Y, t) << endl;

//cout << (-82.0647873) \* t\*t\*t\*t\*t + 240.8373045285334 \* t \* t \* t \* t - 292.282957064 \* t \* t \* t +187.6966416139549 \* t \* t - 67.60908206759223 \* t +12.7584368549594;

}

double lagrang(double \*x, double \*y, double X) {

double sum = 0, p = 1;

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

p = 1;

for (int j = 0; j < n; j++) {

if (i != j) {

p \*= ((X-x[j]) / (x[i] - x[j]));

}

}

sum += y[i] \* p;

}

return sum;}

Листинг программы #2

#include <iostream>

#include <iomanip>

#include <cmath>

#include <algorithm>

using namespace std;

const int n = 11;

void newton(double \*x, double \*y, double \*X);

int Fact(int x);

int main() {

double X[n] = { 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65 };

double Y[n] = { 0.860708, 0.818731, 0.778801, 0.740818, 0.704688, 0.670320, 0.637628, 0.606531, 0.576950, 0.548812, 0.522046 };

double t[4] = { 0.1535, 0.6247, 0.1317,0.6672 };

double e = 0.000001;

cout << "Interpolation according to the Newton formula" << endl;

cout << "X1=0,1535; X2=0,6247; X3=0,1317; X4=0,6672" << endl;

cout << "0,15 0,860708 " << endl

<< "0,20 0,818731 " << endl

<< "0,25 0,778801 " << endl

<< "0,30 0,740818 " << endl

<< "0,35 0,704688 " << endl

<< "0,40 0,670320 " << endl

<< "0,45 0,637628 " << endl

<< "0,50 0,606531 " << endl

<< "0,55 0,576950 " << endl

<< "0,60 0,548812 " << endl

<< "0,65 0,522046 \n\n";

newton(X, Y, t);

}

void newton(double \*x, double \*y, double \*X) {

double a[n][n];

double sum;

for (int i = 0; i < 4; i++) {

sum = 0;

for (int j = 0; j < n; j++) a[j][0] = y[j];

for (int j = 1; j < n; j++)

for (int k = 0; k < n - j; k++)

a[k][j] = a[k + 1][j - 1] - a[k][j - 1];

double q;

if (i % 2 == 0) {

q = (X[i] - x[0]) / 0.05;

sum += y[0];

double p;

for (int j = 1; j < n; j++) {

p = 1;

for (int k = 0; k < j; k++)

p \*= (q - k);

sum += p \* a[0][j] / Fact(j);

}

} else {

q = (X[i] - x[n - 1]) / 0.05;

sum += y[n - 1];

double p;

for (int j = 1; j < n; j++) {

p = 1;

for (int k = 0; k < j; k++)

p \*= (q + k);

//cout << p << " " << a[n - j - 1][j] << endl;

sum += p \* a[n - j - 1][j] / Fact(j);

}

}

cout << "N(X" << i + 1 << ")" << " = " << fixed << setprecision(6) << sum << endl;

}

cout << "\n\n Finite differences: " << endl;

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

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

cout << setw(9) << a[i][j] << " | ";

cout << endl;

}

}

int Fact(int x) {

if (x==0)

return 1;

return x\*Fact(x-1);

}

Листинг программы #3

#include <iostream>

#include <iomanip>

#include <algorithm>

#include <cmath>

using namespace std;

const int n = 10;

const int basis = 2;

int det\_matrix(int (&a)[3][3]) {

return a[0][0] \* (a[1][1] \* a[2][2] - a[2][1] \* a[1][2]) - a[0][1] \* (a[1][0] \* a[2][2] - a[2][0] \* a[1][2]) + a[0][2] \* (a[1][0] \* a[2][1] - a[2][0] \* a[1][1]);

}

int main(int argc, char const \*argv[])

{

int X[n] = { 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 };

int Y[n] = { 95, 116, 139, 163, 190, 219, 250, 283, 319, 355};

cout << "RMS approximation of the function with OLS:" << endl

<< "10 95" << endl

<< "11 116" << endl

<< "12 139" << endl

<< "13 163" << endl

<< "14 190" << endl

<< "15 219" << endl

<< "16 250" << endl

<< "17 283" << endl

<< "18 319" << endl

<< "19 355 \n\n";

double e[7];

double x1 = (X[0] + X[n - 1]) / 2, x2 = sqrt(X[0] \* X[n - 1]), x3 = 2 \* X[0] \* x[n - 1] / (X[0] + X[n - 1]);

double y1 = (Y[0] + Y[n - 1]) / 2, y2 = sqrt(Y[0] \* Y[n - 1]), y3 = 2 \* Y[0] \* Y[n - 1] / (Y[0] + Y[n - 1]);

e[0] =

int matrix[11][7];

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

matrix[i][0] = X[i];

matrix[i][1] = Y[i];

matrix[i][2] = X[i] \* X[i];

matrix[i][3] = X[i] \* X[i] \* X[i];

matrix[i][4] = X[i] \* X[i] \* X[i] \* X[i];

matrix[i][5] = X[i] \* Y[i];

matrix[i][6] = X[i] \* X[i] \* Y[i];

}

for (int i = 0; i < 7; i++) matrix[10][i] = 0;

for (int i = 0; i < 7; i++) {

for (int j = 0; j < n; j++) {

matrix[10][i] += matrix[j][i];

}

}

//cout << matrix[0][1] << endl;

cout << setw(10) << "x" << setw(10) << "y" << setw(10) << "x^2"

<< setw(10) << "x^3" << setw(10) << "x^4" << setw(10) << "x\*y" << setw(10) << "x\*y^2" << endl;

for (int i = 0; i < 11; i++) {

for (int j = 0; j < 7; j++)

cout << setw(9) << matrix[i][j] << " ";

cout << endl;

}

int temp1[3][3] = {{matrix[10][2], matrix[10][0], n},

{matrix[10][3], matrix[10][2], matrix[10][0]},

{matrix[10][4], matrix[10][3], matrix[10][2]}

};

int temp2[3][3] = {{matrix[10][1], matrix[10][0], n},

{matrix[10][5], matrix[10][2], matrix[10][0]},

{matrix[10][6], matrix[10][3], matrix[10][2]}

};

int temp3[3][3] = {{matrix[10][2], matrix[10][1], n},

{matrix[10][3], matrix[10][5], matrix[10][0]},

{matrix[10][4], matrix[10][6], matrix[10][2]}

};

int temp4[3][3] = {{matrix[10][2], matrix[10][0], matrix[10][1]},

{matrix[10][3], matrix[10][2], matrix[10][5]},

{matrix[10][4], matrix[10][3], matrix[10][6]}

};

double a, b, c;

a = double(det\_matrix(temp2))/det\_matrix(temp1);

b = double(det\_matrix(temp3))/det\_matrix(temp1);

c = double(det\_matrix(temp4))/det\_matrix(temp1);

cout << endl << "y = " << a << "x^2 " << b << "x " << c << endl << endl;

double deviation;

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

cout << "x = " << X[i] << setw(7) << " y = " << matrix[i][1] << setw(10) << " y' = " << a \* X[i] \* X[i] + b \* X[i] + c << endl;

deviation += pow(matrix[i][1] - (a \* X[i] \* X[i] + b \* X[i] + c), 2);

}

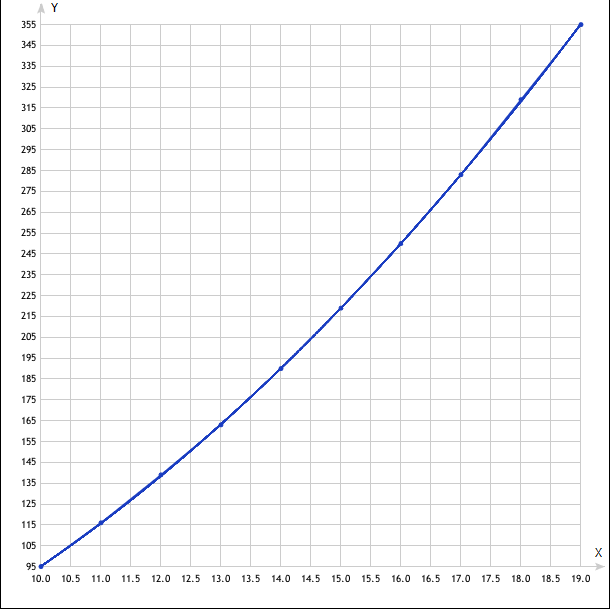
cout << "Standard deviation: " << sqrt(deviation / n);

return 0;

}

Приложение Б

График аппроксимирующей функции #1



Вывод

В ходе данной лабораторной работы были рассмотрены методы приближения функции: интерполяционные многочлены Ньютона и Лагранжа, аппроксимирующие функции среднеквадратического приближения. Были получены знания в построении интерполяционного многочлена Лагранжа по заданной неравномерной сетке с дальнейшим вычислением значения в искомой точке X. Также были применены две формулы интерполяционного многочлена Ньютона: прямая и обратная интерполяционные формулы для заданной равномерной сетки. Для искомых четырёх X были выбраны формулы, исходя из положения X относительно сетки. В качестве аппроксимации значений было изучено среднеквадратическое приближение. Был получен навык в определении зависимости функции, с дальнейшим определением коэффициентов многочлена при помощи метода наименьших квадратов. Так же было посчитано среднеквадратическое отклонение для полученной аппроксимирующей функции, которое оказалось на 2-3 порядка меньше значений функции в таблице, что говорит о довольно точном результате аппроксимирующей функции. В каждой подзадачи ответы совпали с точными с учётом заданной в условии погрешности.