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 Метод обратной матрицы

Метод обратной матрицы используется при решении систем линейных алгебраических уравнений, если число неизвестных равно числу уравнений.

Суть метода обратной матрицы:

Пусть задана система *n* линейных уравнений с *n* неизвестными:

Эту систему можно записать в виде матричного уравнения AX=B, где

*,*

*,*

*.*

Из полученного матричного уравнения необходимо выразить X. Для этого умножим обе части матричного уравнения слева на A-1, получим:

A-1 \*A\*X = A-1 \*B

Так как A-1 \* A = E, то E \* X = A-1 \* B или X = A-1 \* B. Далее находится обратная матрица A-1 и умножается на столбец свободных членов B.

Обратная матрица к матрице А существует только при условии, что detA ≠ 0. Поэтому при решении системы линейных уравнений методом обратной матрицы в первую очередь вычисляется detA. Если detA ≠ 0, то система имеет единственное решение, которое можно получить методом обратной матрицы, если же detA = 0, то методом обратной матрицы решить систему нельзя.

3 Выполнение задания

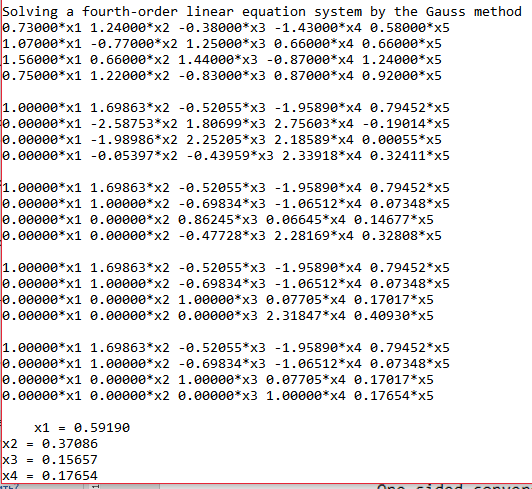


Рисунок 1 – метод Гаусса

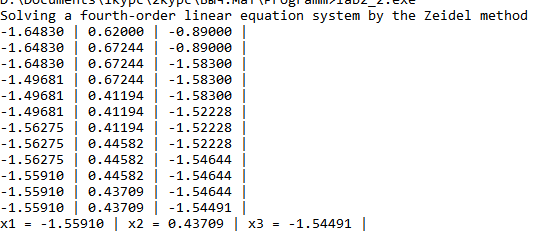


Рисунок 2 –метод Зейделя

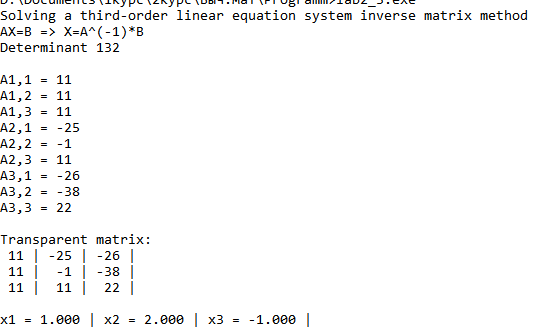


Рисунок 3 –метод обратной матрицы

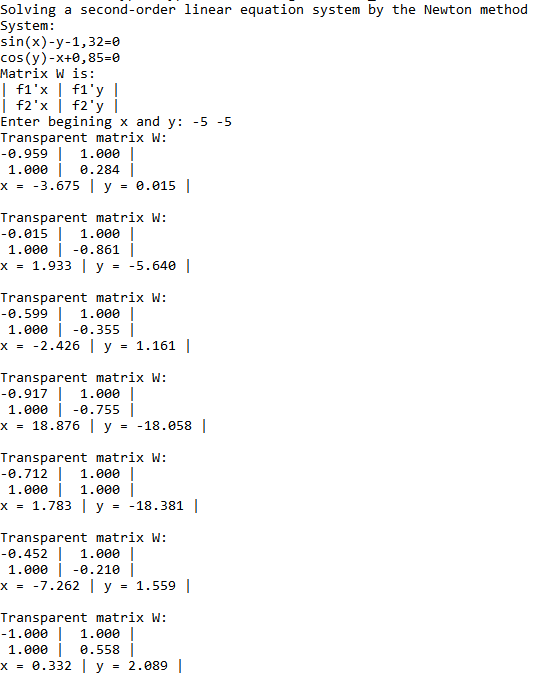


Рисунок 4 –метод Ньютона

Приложение А

Листинг программы #1

#include <iostream>

#include <iomanip>

using namespace std;

double matrix [4][5] = {{0.73, 1.24, -0.38,-1.43, 0.58},

{1.07, -0.77, 1.25, 0.66, 0.66},

{1.56, 0.66, 1.44, -0.87, 1.24},

{0.75, 1.22, -0.83, 0.87, 0.92} };

const int n = 4, m = 5;

void show\_matrix() {

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

for (int j = 0; j < m; j++) cout << fixed << setprecision(5) << matrix[i][j] << "\*x" << j + 1 << " " << setw(5);

cout << endl;

}

cout << endl;

}

int main() {

cout << "Solving a fourth-order linear equation system by the Gauss method" << endl;

double e = 0.001;

show\_matrix();

double tmp, \*xx = new double[m];

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

tmp = matrix[i][i];

for (int j = n; j >= i; j--) matrix[i][j] /= tmp;

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

tmp = matrix[j][i];

for (int k = n; k >= i; k--) matrix[j][k] -= tmp \* matrix[i][k];

}

show\_matrix();

}

xx[n - 1] = matrix[n - 1][n];

for (int i = n - 2; i >= 0; i--) {

xx[i] = matrix[i][n];

for (int j = i + 1; j < n; j++) xx[i] -= matrix[i][j] \* xx[j];

}

for (int i = 0; i < n; i++) cout << "x" << i + 1 << " = " << xx[i] << endl;

delete [] xx;

}

Листинг программы #2

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

const int n = 3;

const double e = 0.01;

bool converge(double \*xk, double \*xkp) {

double norm = 0;

for (int i = 0; i < n; i++) norm += (xk[i] - xkp[i]) \* (xk[i] - xkp[i]);

return (sqrt(norm) < e);

}

int main() {

cout << "Solving a fourth-order linear equation system by the Zeidel method" << endl;

double A [4][4] = { {0.17, 0.31, -0.18, 0.22},

{-0.21, 0, 0.33, 0.22},

{0.32, -0.18, 0.05, -0.19},

{0.12, 0.28, -0.14, 0}

};

double B[4] = {-1.71, 0.62, -0.89, 0.94};

double x[4], p[4];

for (int i = 0; i < n; i++) x[i] = B[i];

do {

for (int i = 0; i < n; i++) p[i] = x[i];

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

double var = 0;

for (int j = 0; j < n; j++) var += (A[i][j] \* x[j]);

x[i] = B[i] + var;

for (int i = 0; i < n; i++) cout << setw(5) << fixed << setprecision(5) << x[i] << " | ";

cout << endl;

}

}

while (!converge(x, p));

for (int i = 0; i < n; i++) cout << "x" << i + 1 << " = " << x[i] << " | ";

}

Листинг программы #3

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

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 \_A(int i, int j, int (&a)[3][3]) {

int a2[2][2];

int count = 0;

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

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

if (k != i && l != j) {

a2[count / 2][count % 2] = a[k][l];

count++;}

return pow(-1, i + j) \* (a2[0][0] \* a2[1][1] - a2[0][1] \* a2[1][0]);

}

int main() {

cout << "Solving a third-order linear equation system inverse matrix method" << endl << "AX=B => X=A^(-1)\*B" << endl;

int n = 3;

int A[3][3] = { {3, 2, 7},

{-5, 4, 1},

{1, -3, 2}

};

int B[3] = {0, 2, -7};

double X[3];

int det;

int A\_T[3][3];

if (det\_matrix(A) != 0) {

det = det\_matrix(A);

cout << "Determinant " << det << endl << endl;

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

for (int j = 0; j < 3; j++) {

A\_T[j][i] = \_A(i, j, A);

cout << "A" << i + 1 << "," << j + 1 << " = " << A\_T[j][i] << endl;

}

cout << endl << "Transparent matrix: " << endl;

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

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

cout << setw(3) << A\_T[i][j] << " | ";

cout << endl;

}

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

int val = 0;

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

val += A\_T[i][j] \* B[j];

X[i] = 1 / double(det) \* val;

}

cout << endl;

for (int i = 0; i < 3; i++) cout << "x" << i + 1 << " = " << fixed << setprecision(3) << X[i] << " | ";

} else cout << "The system has no solutions.";

}

Листинг программы #4

#include <iostream>

#include <iomanip>

#include <cmath>

#include <algorithm>

using namespace std;

double f1(double x, double y) {

return sin(x) - y - 1.32;}

double f2(double x, double y) {

return cos(y) - x + 0.85;}

double f1\_x(double x, double y) {

return cos(x);}

double f1\_y(double x, double y) {

return -1;}

double f2\_x(double x, double y) {

return -1;}

double f2\_y(double x, double y) {

return -sin(y);}

int main() {

cout << "Solving a second-order linear equation system by the Newton method" << endl;

double e = 0.001;

cout << "System: " << endl

<< "sin(x)-y-1,32=0" << endl

<< "cos(y)-x+0,85=0" << endl;

cout << "Matrix W is: " << endl

<< "| f1'x | f1'y |" << endl

<< "| f2'x | f2'y |" << endl;

double x, y, W[2][2], dx, dy, norm;

cout << "Enter begining x and y: ";

cin >> x >> y;

do {

W[0][0] = f2\_y(x, y);

W[0][1] = -f1\_y(x, y);

W[1][0] = -f2\_x(x, y);

W[1][1] = f1\_x(x, y);

cout << "Transparent matrix W: " << endl;

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++)

cout << setw(6) << fixed << setprecision(3) << W[i][j] << " | ";

cout << endl;

}

double det = W[0][0] \* W[1][1] - W[0][1] \* W[1][0];

dx = (W[0][0] \* f1(x, y) + W[0][1] \* f2(x, y)) / det;

dy = (W[1][0] \* f1(x, y) + W[1][1] \* f2(x, y)) / det;

double px = x, py = y;

x -= dx;

y -= dy;

cout << "x = " << x << " | y = " << y << " |" << "\n\n";

norm = max((px - x) \* (px - x), (py - y) \* (py - y));

} while (norm >= e);

cout << "x = " << x << " | y = " << y << " |";

}