|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
| МИНОБРНАУКИ РОССИИ | | | |
| 1. Федеральное государственное бюджетное образовательное учреждение 2. высшего профессионального образования 3. **"Московский технологический университет"** 4. МИРЭА | | | |
| Институт Кибернетики | | |  |
| Кафедра программного обеспечения систем радиоэлектронной аппаратуры при АО «Концерн радиостроения «ВЕГА» | | |  |

|  |  |
| --- | --- |
| **Отчет по лабораторной работе №3** | |
| **по дисциплине** | |
| **«** Численные методы **»** | |
| **Вариант 7** | |
| Студент 3-го курса  группы КМБО-2-16 | Миронов Д.А. |
| Преподаватель | Даева С.Г. |
| Рецензент |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2018 г. |  |
|  |  |  |
| «Допущен к защите» | «\_\_»\_\_\_\_\_\_\_2018 г. |  |

Москва 2018

Содержание

Задание 3

Теоретическая часть 3

Практическая часть 4

Приложения 6

# Задание

Решить систему нелинейных уравнений с точностью до 0.001 методами:

1) Ньютона(модифицированный)

2) Градиентный

## Теоретическая часть

Пусть дана система ***n***нелинейныхуравнений с ***n***неизвестными. Общий вид системы:

или где - нелинейные функции.

Требуется найти такой вектор который при подстановки в систему превращает каждое уравнение в верное числовое равенство.

Все функции непрерывны и соответственно дифференцируемы по всем своим неизвестным в некоторой выпуклой области существования неизвестной. Под выпуклой областью понимается такая область, в которой производная по каждой неизвестной не меняет свои знаки.

**«Модифицированный метод Ньютона».**

Формула для нахождения решения:

где -матрица Якоби в начальном приближении **x(0)**

причем а - обратная матрица Якоби в начальном приближении **x(0)** соответственно.

Матрица не изменяется от итерации к итерации.

Окончание итерационного процесса произойдет при достижении точности и при условии:

**«Метод градиентного спуска».**

Формула для нахождения решения: .

Где

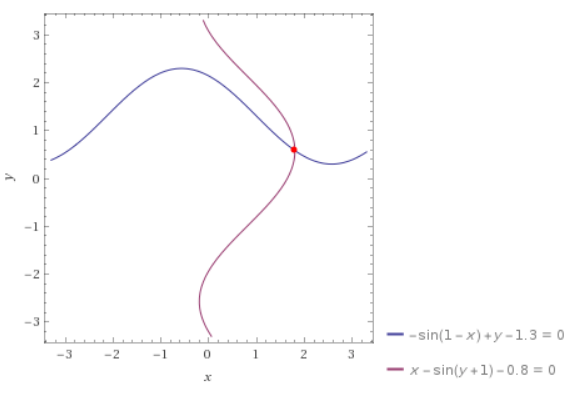
– транспонированная матрица Якоби, а = – вектор результатов функций в приближении .

**Практическая часть**

Для решения нелинейных систем уравнений методами «модифицированный методом Ньютона» и «метод градиентного спуска» были разработаны программы «lab3.1» и «lab3.2» соответственно. Программа написана на языке С++ в операционной системе «ubuntu» с использованием компилятора «GNU GCC».

**1)** Решим систему

модифицированным методом Ньютона.

Графически уточним корни:

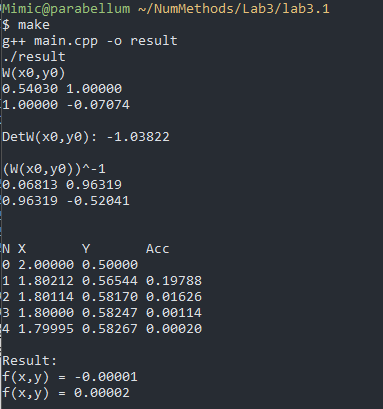
Вычислим матрицу Якоби для нашей системы:

В качестве начального приближения возьмем:

Определитель отличен от нуля, значит существует обратная матрица.

Теперь запишем обратную матрицу Якоби в начальном приближении :

Организуем итерационный процесс:

Вывод программы:

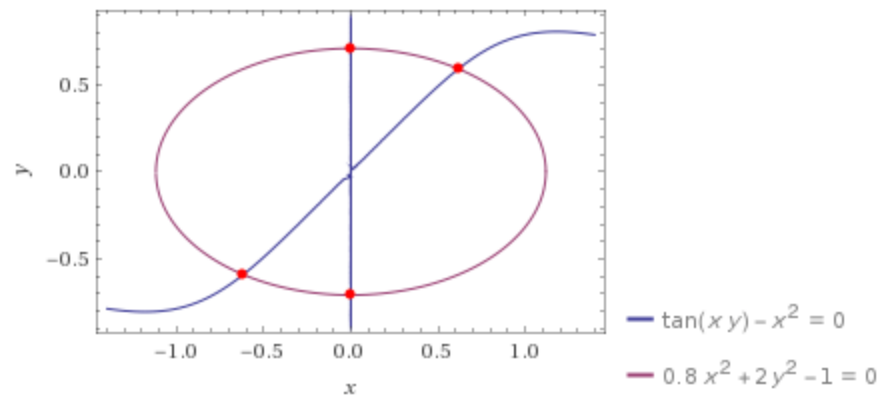
**Ответ:**

**2)** Решим систему

Методом градиентного спуска.

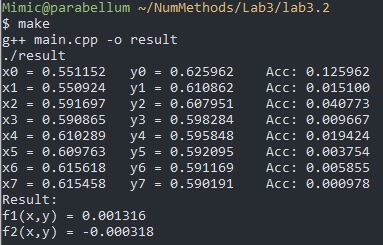
Вычислим матрицу Якоби для нашей системы:

Графически уточним корни:



В качестве начального приближения возьмем:

Организуем итерационный процесс .

Вывод программы:

**Ответ:**

**Приложения**

*Код программы «lab3.1»*

#include <iostream>

#include <cmath>

#include <stdlib.h>

#include <iomanip>

using namespace std;

double First(double x1, double x2)

{

return sin(x1-1)-1.3+x2;

}

double Second(double x1, double x2)

{

return x1-sin(x2+1)-0.8;

}

double derX1First(double x1, double x2)

{

return cos(1-x1);

}

double derX2First(double x1, double x2)

{

return 1;

}

double derX1Second(double x1)

{

return 1;

}

double derX2Second(double x2)

{

return -cos(x2+1);

}

int main()

{

const int SIZE = 2;

long double vec[SIZE];

vec[0] = 2;

vec[1] = 0.5;

double matrix[SIZE][SIZE];

cout.setf(ios::fixed);

cout.precision(5);

matrix[0][0] = derX1First(vec[0], vec[1]);

matrix[0][1] = derX2First(vec[0], vec[1]);

matrix[1][0] = derX1Second(vec[0]);

matrix[1][1] = derX2Second(vec[1]);

cout << "W(x0,y0)" << endl;

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

{

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

{

cout << setw(7) << matrix[i][j] << " ";

}

cout << endl;

}

cout << endl;

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

cout << "DetW(x0,y0): " << det << endl;

double temp1 = matrix[0][0];

matrix[0][0] = matrix[1][1];

matrix[1][1] = temp1;

matrix[0][1] = -matrix[0][1];

matrix[1][0] = -matrix[1][0];

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

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

{

matrix[i][j] = (1 / det)\*matrix[i][j];

}

cout << endl;

cout << "(W(x0,y0))^-1" << endl;

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

{

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

{

cout << setw(7) << matrix[i][j] << " ";

}

cout << endl;

}

double F[2];

F[0] = First(vec[0], vec[1]);

F[1] = Second(vec[0], vec[1]);

cout << endl << endl;

double iterArr[100][SIZE];

for (int i = 0; i < 100; ++i)

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

{

iterArr[i][j] = 0;

}

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

{

iterArr[0][i] = vec[i];

}

cout << "N " << "X\t " << "Y\t " << "Acc" << endl;

cout << "0 " << iterArr[0][0] << " " << iterArr[0][1] << endl;

int n = 0;

double max = 0;

for (int c = 1;; c++)

{

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

{

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

{

iterArr[c][i] += matrix[i][j] \* F[j];

}

iterArr[c][i] = iterArr[c - 1][i] - iterArr[c][i];

vec[i] = iterArr[c][i];

double k = fabs(iterArr[c][i] - iterArr[c - 1][i]);

if (k > max) max = k;

if (i == 0) cout << c << " ";

cout << setw(6) << iterArr[c][i] << " ";

}

F[n] = First(vec[0], vec[1]);

F[++n] = Second(vec[0], vec[1]);

n = 0;

cout << setw(6) << max;

if (max < 0.001)

{

cout << "\n\nResult: " << endl;

cout << "f(x,y) = " << First(vec[0], vec[1]) << endl;

cout << "f(x,y) = " << Second(vec[0], vec[1]) << endl;

break;

}

else max = 0;

cout << endl;

}

return 0;

}

*Код программы «lab3.2»*

#include <iostream>

#include <cmath>

#include <math.h>

#include <stdlib.h>

#include <iomanip>

using namespace std;

float res[2] = {0.5, 0.5};

float resn[2];

float m = 0;

float eps = 0.001;

float f1(float x, float y)

{

float res;

res = tan(x\*y)-x\*x;

return res;

}

float f2(float x, float y)

{

float res;

res = 0.8\*x\*x+2\*y\*y-1;;

return res;

}

float f1dx(float x, float y)

{

float res;

res = y/(cos(x\*y)\*cos(x\*y)) - 2\*x;

return res;

}

float f1dy(float x, float y)

{

float res;

res = x/(cos(x\*y)\*cos(x\*y));

return res;

}

float f2dx(float x, float y)

{

float res;

res = 1.6\*x;

return res;

}

float f2dy(float x, float y)

{

float res;

res = 4\*y;

return res;

}

void count\_m(float x, float y)

{

float wf1;

float wf2;

wf1 = (f1dx(x,y)\*f1dx(x,y)+f1dy(x,y)\*f1dy(x,y))\*f1(x,y)+

(f1dx(x,y)\*f2dx(x,y)+f1dy(x,y)\*f2dy(x,y))\*f2(x,y);

wf2 = (f1dx(x,y)\*f2dx(x,y)+f1dy(x,y)\*f2dy(x,y))\*f1(x,y) +

(f2dx(x,y)\*f2dx(x,y)+f2dy(x,y)\*f2dy(x,y))\*f2(x,y);

m = (f1(x,y)\*wf1+f2(x,y)\*wf2)/(wf1\*wf1+wf2\*wf2);

}

int main()

{

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

{

count\_m(res[0], res[1]);

resn[0] = res[0]-m\*(f1dx(res[0],res[1])\*f1(res[0],res[1])+f2dx(res[0],res[1])\*f2(res[0],res[1]));

resn[1] = res[1]-m\*(f1dy(res[0],res[1])\*f1(res[0],res[1])+f2dy(res[0],res[1])\*f2(res[0],res[1]));

float max\_eps = (fabs(res[0]-resn[0])>fabs(res[1]-resn[1]))?fabs(res[0]-resn[0]):fabs(res[1]-resn[1]);

res[0]=resn[0];

res[1]=resn[1];

printf("x%d = %f\ty%d = %f\t Acc: %f\n", i, res[0], i, res[1], max\_eps);

if(max\_eps<eps)

break;

}

printf("Result: \nf1(x,y) = %f\nf2(x,y) = %f\n\n", f1(res[0],res[1]), f2(res[0],res[1]));

return 0;

}