Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ**

**Отчет по лабораторной работе №4**

**По дисциплине**

**«Численные методы»**

Студентка гр. 430-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Лузинсан

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Проверил: ст. преп. каф. АСУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Е. Косова

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

Томск 2022

Оглавление

[1 Цели и задачи 3](#_Toc102396828)

[1.1 Формат входных данных 3](#_Toc102396829)

[1.2 Формат выходных данных 3](#_Toc102396830)

[2 Теория 4](#_Toc102396831)

[2.1 Методы решения 5](#_Toc102396832)

[2.1.1 Метод Ньютона 5](#_Toc102396833)

[2.1.2 Метод итераций 5](#_Toc102396834)

[2.1.3 Метод наискорейшего спуска 5](#_Toc102396835)

[3 Ход выполнения лабораторной работы 7](#_Toc102396836)

[Тестирование 9](#_Toc102396837)

[3.1 Unit 1 9](#_Toc102396838)

[3.2 Unit 2 9](#_Toc102396839)

[3.3 Unit 3 10](#_Toc102396840)

[Вывод 11](#_Toc102396841)

[Список использованных источников 12](#_Toc102396842)

[**Листинг программы** 13](#_Toc102396843)

# Цели и задачи

В ходе данной лабораторной работы необходимо реализовать поиск решения системы нелинейных уравнений с помощью методов Ньютона, итераций и наискорейшего спуска.

## 1.1 Формат входных данных

|  |  |
| --- | --- |
| m | – метод; |
| n  x0 | – размерность СНУ;  – начальное приближение;  – требуемая погрешность решения; |
| … | – система функций. |

## 1.2 Формат выходных данных

|  |  |
| --- | --- |
| …  \* | – последовательные приближения решения СНУ;  – вектор невязки ; |
| \* | – норма вектора невязки. |

# Теория

Не всегда системы уравнений, которые приходится решать в различных задачах, бывают линейными. Для решения систем нелинейных уравнений (СНУ) существует ряд специальных методов для их решения. По аналогии с решением уравнений с одной переменной, можно заключить, что численные методы позволяют быстрее получить приближенное решение при помощи ЭВМ. А также СНУ большой размерности аналитически очень тяжело решаются.

В матричном виде СНУ выглядит следующим образом:

|  |  |
| --- | --- |
| где , т.е. | (2.1) |

Если n<m, то система может иметь множество решений. Если n>m, то система переопределена. В этом случае у неё может не быть решений. Мы будем рассматривать ситуацию с n = m. В этом случае количество решений зависит от вида системы функций F. Какое именно решение будет найдено, зависит от начальной точки x0.

Очевидно, что при n=m=1 получим обычное уравнение с одной переменной. В принципе, все рассмотренные методы в таком случае вырождаются в методы решения уравнений с одной переменной. Аналогией производной при n ≠ 1 выступает матрица Якоби:

|  |  |
| --- | --- |
|  | (2.2) |

При n = 1 якобиан вырождается в обычную производную.

## Методы решения

### **Метод Ньютона**

Итерационный процесс, по аналогии с формулами метода Ньютона для решения уравнений с одной переменной, выглядит следующим образом:

|  |  |
| --- | --- |
|  | (2.3) |
| где *Ф* | (2.4) |

Критерий окончания итерационного процесса, выглядит так:

|  |  |
| --- | --- |
|  | (2.5) |

### Метод итераций

Как и метод Ньютона, метод итераций решения СНУ является обобщением метода итераций решения уравнений с одной переменной и имеет вид (2.3). Причём для повышения скорости сходимости матрицу Якоби в (2.4) нужно вычислять не в точке x(k), а в некоторой другой точке. Очевидно, что в данном случае определить её гораздо труднее. Поэтому обычно просто берут точку x(0):

|  |  |
| --- | --- |
|  | (2.6) |

В итоге получаем модифицированный метод Ньютона, и скорость сходимости только падает. Критерий останова определяется выражением (2.5).

### Метод наискорейшего спуска

Итерационный процесс строится по общей формуле (2.3), где:

|  |  |
| --- | --- |
|  | (2.7) |

Функция *U(x)* преобразует систему функций *f* в скалярную функцию векторного аргумента:

|  |  |
| --- | --- |
|  | (2.8) |

Очевидно, что:

|  |  |
| --- | --- |
|  | (2.9) |

Т.е. единственной проблемой остается поиск параметра . Он должен минимизировать функцию *Ф(x)* вдоль направления *∇U(x)*:

|  |  |
| --- | --- |
|  | (2.10) |

Очевидно, что он должен быть положительным, иначе мы будем двигаться в направлении градиента, а не антиградиента функции (т.е. искать максимум).

Как известно, в точке минимума (как и в других точках экстремума) значение производной функции равно нулю. Используем этот факт для минимизации выражения (2.10):

|  |  |
| --- | --- |
|  | (2.11) |

Уравнение (2.11) можно решить численно, если использовать правила дифференцирования. Можно его решить и аналитически, если прибегнуть к некоторым приближениям. Тогда получим:

|  |  |
| --- | --- |
|  | (2.12) |

где

# Ход выполнения лабораторной работы

В ходе данной лабораторной работы был реализован класс NonlinearEquations, в котором содержатся основные методы Ньютона Newton() и наискорейшего спуска SteepestDescent().

В процессе реализации метода Ньютона было выяснено, что в некоторый момент вычисления матрицы Якоби, могут возникать неопределённые значение, в связи с чем было добавлено условие, что если такой случай имел место быть, то матрица Якоби остаётся от предыдущей итерации. Таким образом это явилось неким аналогом метода итераций, в связи с чем сам метод итераций явно реализован не был.

Метод Ньютона начинается с получения матрицы Якоби. В случае её успешного вычисления, рассчитывается следующее приближенное решение системы по формуле (2.4). Для того чтобы использовать эту формулу, вызываются функции получения обратной матрицы (методом Гаусса) getInverseMatrixByMethod() и получения вектор-столбца значений системы функций при заданных аргументах getFunctionValues(). Итерационный процесс прекращается, как только модуль разницы текущего и предыдущего приближения станет меньше, чем требуемая погрешность, либо когда максимальное количество итераций было превышено.

Метод Наискорейшего спуска похож по реализации с методом Итераций (Ньютона). После вычисления матрицы Якоби, если всё прошло успешно, вычисляется вектор-столбца значений системы функций при заданных аргументах, так как он нам понадобится ещё неоднократно. В момент получения следующего приближения используется формула (2.7), где требуется найти параметр минимизации шага спуска (метод getMinimizingValue()) и вектор-столбец направления градиента. Для нахождения направления градиента используются уже известные данные в формуле (2.9). В методе получения параметра минимизации определяются переменные транспонированной матрицы Якоби, вектора градиента и транспонированного вектора градиента. А далее по формуле 2.12 получается нужный параметр. Итерационный процесс прекращается, как только модуль разницы текущего и предыдущего приближения станет меньше, чем требуемая погрешность, либо когда максимальное количество итераций было превышено.

# Тестирование

## Unit 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| СНУ | x0 | eps0 | Метод | Итераций (k) | (xk) | Вектор невязки (e\*) | Норма вектора невязки (||e\*||) |
|  |  | 1e-3 | Ньютона | 6 |  |  | 0.000898747 |
| Наискорейшего спуска | 37 |  |  | 0.000984548 |

## Unit 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| СНУ | x0 | eps0 | Метод | Итераций (k) | (xk) | Вектор невязки (e\*) | Норма вектора невязки (||e\*||) |
|  |  | 1e-5 | Ньютона | 3 |  |  | 3.94483e-07 |
| Наискорейшего спуска | 100000 (MAX) | (не сходится) |  | 0.163283 |

## Unit 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| СНУ | x0 | eps0 | Метод | Итераций (k) | (xk) | Вектор невязки (e\*) | Норма вектора невязки (||e\*||) |
|  |  | 5e-3 | Ньютона | 3 |  |  | 0.00460618 |
| Наискорейшего спуска | 30 |  |  | 0.00438926 |

# Вывод

В ходе данной лабораторной работы я реализовала поиск решения системы нелинейных уравнений с помощью методов Ньютона, итераций и наискорейшего спуска.

# Список использованных источников

1. Вычислительные методы: учебное пособие / А.А.Мицель. — Томск: Эль Контент, 2013. — 198 с. (дата обращения: 02.05.2022).

**Листинг программы**

#pragma once

#include <fstream>

#include <string>

#include <iomanip>

#include "../../../2/Lab2Chm/Lab2Chm/Matrix.h"

#include "../../../2/Lab2Chm/Lab2Chm/Vector.h"

#include "PolStr.h"

#define MAX\_ITER 100000

namespace luMath

{

template<class T>

class NonlinearEquations

{

public:

enum class METHOD

{

NEWTON=1,

STEEPESTDESCENT,

};

private:

METHOD \_method; // Требуемый метод решения СНУ

int n; // Размерность СНУ

Vector<T> x0; // Начальное приближение

T eps; // Требуемая погрешность решения

Vector<std::string> FunSys; // Система функций

std::ofstream \_fout; // Выходной файл

public:

NonlinearEquations()

{

std::ifstream \_fin("input.txt");

\_fout = std::ofstream("output.txt");

T c;

\_fin >> c >> n;

\_method = static\_cast<METHOD>(c);

x0 = Vector<T>(n);

x0.transposition();

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

{

\_fin >> c;

x0[i] = c;

}

\_fin >> eps;

FunSys = Vector<std::string>(n);

\_fin.seekg(2, std::ios\_base::cur);

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

getline(\_fin, FunSys[i]);

FunSys.transposition();

\_fout << "\n\tСистема Нелинейных Уравнений:\n" << std::setw(5) << FunSys << '\n';

\_fin.close();

}

~NonlinearEquations()

{

\_fout.close();

}

METHOD getMethod() { return \_method; }

static Vector<T> GaussMethod(const Matrix<T>& A, const Vector<T>& b)

{

Matrix<T> expandedMatrix(A.getRows(), A.getCols() + 1);

for (int i = 0; i < expandedMatrix.getRows(); i++)

for (int j = 0; j < expandedMatrix.getCols(); j++)

if (j == expandedMatrix.getCols() - 1)

expandedMatrix[i][j] = b[i];

else

expandedMatrix[i][j] = A[i][j];

return GaussMethod(expandedMatrix);

}

static Vector<T> GaussMethod(const Matrix<T>& expandedMatrix)

{

Matrix<T> tempMatrix(expandedMatrix);

for (int i = 0; i < tempMatrix.getRows(); i++)

{

T coeff = tempMatrix[i][i];

for (int j = i; j < tempMatrix.getRows() + 1; j++)

tempMatrix[i][j] /= coeff;

for (int j = i + 1; j < tempMatrix.getRows(); j++)

{

coeff = tempMatrix[j][i];

for (int k = i; k < tempMatrix.getCols(); k++)

tempMatrix[j][k] -= coeff \* tempMatrix[i][k];

}

}

Vector<T> result(tempMatrix.getRows());

result.transposition();

for (int i = result.getLength() - 1; i >= 0; i--)

{

T sumCoeff = 0;

for (int j = i + 1; j < result.getLength(); j++)

sumCoeff += tempMatrix[i][j] \* result[j];

result[i] = tempMatrix[i][result.getLength()] - sumCoeff;

}

return result;

}

static Matrix<T> getInverseMatrixByMethod(Vector<T>(\*Method)(const Matrix<T>&, const Vector<T>&), Matrix<T> matrix)

{

int m = matrix.getCols();

Matrix<T> inverseMatrix(m);

Vector<Vector<T>> x\_temp(m);

Vector<Vector<T>> E(m);

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

{

E[i] = Vector<T>(m);

E[i][i] = 1;

E[i].transposition();

}

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

x\_temp[i] = Method(matrix, E[i]);

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

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

inverseMatrix[i][j] = x\_temp[j][i];

return inverseMatrix;

}

static Matrix<T>\* getJacobi(Vector<std::string> FunSys, Vector<T> x)

{

int n = x.getLength();

Matrix<T> \*J = new Matrix<T>(n);

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

{

const char\* polStr = CreatePolStr(FunSys[i].c\_str(), n);

if (GetError() == ERR\_OK)

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

{

const double\* x\_p = x.getPointer();

(\*J)[i][j] = EvalPolStr(polStr, x\_p, 1, j + 1);

if (isnan((\*J)[i][j])) return NULL;

}

else std::cerr << "Error: " << GetError();

}

return J;

}

// Вектор-столбец значений системы функций при заданных аргументах

static Vector<T> getFunctionValues(Vector<std::string> FunSys, Vector<T> arg)

{

int n = FunSys.getLength();

Vector<T> f\_x(n);

f\_x.transposition();

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

{

const char\* polStr = CreatePolStr(FunSys[i].c\_str(), n);

if (GetError() == ERR\_OK)

{

const double\* arg\_p = arg.getPointer();

f\_x[i] = EvalPolStr(polStr, arg\_p);

}

else std::cerr << "Error: " << GetError();

}

return f\_x;

}

// Вектор столбец направления градиента

// при заданной матрице Якоби, вектор-столбце значений системы функций при текущем приближении и сам вектол-столбец аргументов приближения

static Vector<T> getGradientDirection(const Matrix<T>& Jacobi, const Vector<T>& FunctionValues, const Vector<T>& x0)

{

Matrix<T> Jacobi\_T(Jacobi);

Jacobi\_T.transposition();

return 2 \* Jacobi\_T \* x0;

}

// Поиск параметра 'lambda'

static const T& getMinimizingValue(const Matrix<T>& Jacobi, const Vector<T>& FunctionValues, const Vector<T>& x0)

{

Matrix<T> Jacobi\_T(Jacobi);

Jacobi\_T.transposition();

Vector<T> g(Jacobi\_T \* FunctionValues);

Vector<T> g\_T(g);

g\_T.transposition();

return \*((g\_T \* g).getPointer())

/ \*((2 \* g\_T \* Jacobi\_T \* Jacobi \* g).getPointer());

}

// Метод Ньютона (модицицированный)

Vector<T> Newton()

{

\_fout << "\n\tМетод Ньютона:\n\t";

Vector<T> x1(x0);

int index = 0;

\_fout << "x^[" << index << "]:\n" << x1 << "\n";

Matrix<T>\* J = new Matrix<T>(n), \*tempJ;

bool flag = true;

do {

index++;

x0 = x1;

if (flag)

{

tempJ = getJacobi(FunSys, x0);

if (tempJ) J = tempJ;

else flag = false;

}

x1 = x0

- getInverseMatrixByMethod(NonlinearEquations::GaussMethod, \*J)

\* getFunctionValues(FunSys, x0);

\_fout.width(10);

printIter(\_fout, x1, index);

} while ((x1-x0).getModule() >= eps && index < MAX\_ITER);

delete[] J, tempJ;

return x1;

}

// Метод Наискорейшего спуска

Vector<T> SteepestDescent()

{

\_fout << "\n\tМетод Наискорейшего спуска:\n\t";

Vector<T> x1(x0), f\_x;

int index = 0;

\_fout << "x^[" << index << "]:\n" << x1 << "\n";

Matrix<T>\* J = new Matrix<T>(n), \* tempJ;

bool flag = true;

do {

index++;

x0 = x1;

if (flag)

{

tempJ = getJacobi(FunSys, x0);

if (tempJ) J = tempJ;

else flag = false;

}

f\_x = Vector<T>(getFunctionValues(FunSys, x0));

x1 = x0

- getMinimizingValue(\*J, f\_x, x0)

\* getGradientDirection(\*J, f\_x, x0);

\_fout.width(10);

printIter(\_fout, x1, index);

} while ((x1 - x0).getModule() >= eps && index < MAX\_ITER);

delete J, tempJ;

return x1;

}

std::ostream& printIter(std::ostream& out, const Vector<T>& x1, int index)

{

std::streamsize width = out.width(), precision = out.precision();

if (!width) width = 5;

out << "\tx^[" << index << "]:\n" << std::setw(width) << std::setprecision(precision) << x1

<< "\t\t\t\t\t\t\tПогрешность \n\tВектор невязки:\n"

<< std::setw(width) << std::setprecision(precision) << (x1 - x0)

<< "\tНорма вектора невязки:\t"

<< (x1 - x0).getModule() << "\n\_\t\_\t\_\t\_\t\_\t\_\t\_\t\_\t\_\t\_\t\_\t\n\n";

return out;

}

};

}