**Министерство науки и высшего образования РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Уфимский университет науки и технологий»**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 100 |  |  |  |  |  |  |  |  |  |  |
| 90 |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |

**СРАВНЕНИЕ ДЕТЕРМИНИРОВАННЫХ МЕТОДОВ ДЛЯ ИНВЕРСИИ ДАННЫХ ЭЛЕКТРОМАГНИТНОГО КАРОТАЖА**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе по дисциплине

«Численные методы»

**3952.335102.000 ПЗ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа  ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмурзин М.Э. |  |  |  |
| Консультант | Касаткин А.А. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

Уфа 2023

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Кафедра Высокопроизводительных вычислительных технологий и систем

**ЗАДАНИЕ**

на курсовую работу по дисциплине

**«Численные методы»**

Студент: Акмурзин Михаил Эдуардович Группа: ПМ-357

Консультант: Касаткин Алексей Александрович

1. Тема курсовой работы

Сравнение детерминированных методов для инверсии данных электромагнитного каротажа

2. Основное содержание

2.1. Изучить детерминированные методы инверсии, основанные на минимизации функции.

2.2. Реализовать метод Ньютона-Гаусса и Нелдера-Мида для выбора оптимальных параметров модели (при отклонении которых будет минимальное отклонение измерений от заданного вектора значений).

2.3. Провести вычислительные эксперименты, сравнить эффективность работы методов.

2.4. Оформить пояснительную записку.

3. Требования к оформлению материалов работы

3.1. Требования к оформлению пояснительной записки

Пояснительная записка к курсовой работе должна быть оформлена в соответствии с требованиями ГОСТ и содержать

• титульный лист,

• задание на курсовую работу,

• содержание,

• введение,

• заключение,

• список литературы,

• приложение, содержащее листинг разработанной программы, если таковая имеется.

|  |  |
| --- | --- |
| Дата выдачи задания  "\_\_" \_\_\_\_\_\_\_\_ 202\_ г. | Дата окончания работы  "\_\_" \_\_\_\_\_\_\_\_\_\_ 202\_ г. |

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Касаткин А.А.

**Содержание**

[Введение 5](#_Toc159180922)

[1. Теоретическая часть 6](#_Toc159180923)

[1.1. Метод нелдера-мида 6](#_Toc159180924)

[1.2. Метод ньютона-гаусса 7](#_Toc159180925)

[2. Практическая часть 9](#_Toc159180926)

[Заключение 10](#_Toc159180927)

[Список литературы 11](#_Toc159180928)

# ВВЕДЕНИЕ

В современной геофизике и геологии электромагнитный каротаж играет важную роль в изучении свойств подземных пород. Однако, интерпретация данных, полученных в ходе разведки, сложна для обработки. Для решения этой проблемы используются различные методы оптимизации.

Целью данного исследования является реализация и сравнение детерминированных методов для инверсии данных электромагнитного каротажа.

В рамках курсовой работы были решены следующие задачи:

1. Изучены методы Ньютона-Гаусса и Нелдера-Мида.
2. Реализованы программно на языке программирования C++ методы Ньютона-Гаусса и Нелдера-Мида.
3. Проведены испытания на реальных данных для сравнения методов Ньютона-Гаусса и Нелдера-Мида.

Результаты этого исследования могут быть полезны для специалистов в области геофизики и геологии, а также для разработчиков программного обеспечения, работающих над реализацией и оптимизацией алгоритмов для обработки и интерпретации геофизических данных.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Метод Нелдера-Мида

Метод Нелдера-Мида — это метод оптимизации, который не требует вычисления градиентов функции. Он основан на концепции симплекса, который является многогранником в n-мерном пространстве.

Пусть у нас есть функция:

Мы хотим найти , такое что для всех .

Выберем точки , так чтобы

, где – орты, - произвольная точка, , , - расстояние между точками. Множество равноудалённых точек называется регулярный симплекс.

Найдем в каждой точке значения функции и упорядочим

Обозначим. Найдем центр тяжести всех точек, за исключением точки , в которой достигается наибольшее значение функции (1). Пусть центром тяжести будет точка

Вычислим . Удобнее начать перемещение от точки от точки . Отразим точку относительно , получим точку

, где – коэффициент отражения. Обозначим.

Сравним значения функции .

В случае если , то мы получили наименьшее значение функции. Направление из точки  в точку наиболее удобно для перемещения. Таким образом, мы производим растяжение в этом направление и находим точку

, где – коэффициент растяжения. Обозначим. Если , то заменяем на точку и переходим к проверке сходимости (9). Иначе отбрасываем точку . Очевидно, что мы переместились слишком далеко от точки центра масс к точке . Поэтому и переходим к проверке сходимости (9).

В случае если сравним точки . Если , то

и переходим к проверке сходимости (9). Иначе если , то

и переходим к проверке сходимости (9). Если , то переходим к сжиманию многогранника, т.е. находим точку

, где – коэффициент сжатия. Обозначим. Сравним функции . Если , то и переходим к проверке сходимости (9). Иначе мы уменьшаем размер симплекса делением пополам расстояние от каждой точки симплекса до точки, определяющей наименьшее значение функции.

, где . Далее переходим к проверке сходимости (9).

Проверка сходимости основа на том, чтобы стандартное отклонение – го значения функции было меньше некоторого заданного малого значения ε. В этом случае вычисляется

Критерий окончания поиска состоит в проверке условия , где ε – произвольное малое число, а – значение функции в центре тяжести. Для продолжения поиска перейдем к шагу (3) и продолжим вычисления до тех пор, пока .

Коэффициенты являются соответственно коэффициентами отражения, сжатия и растяжения. В качестве удовлетворяющих значения этих параметров при оптимизации без ограничений Нелдер и Мид рекомендовали .

## Метод Ньютона-Гаусса

Пусть является нелинейной дважды дифференцируемой функцией из для любого Для данного

рассмотрим нелинейную задачу подбора данных методом наименьших квадратов:

, где вектор-функция невязки между математический ожиданием модели и вектором данных *b*.

Перепишем задачу минимизации (10):

Пусть якобиан остаточной вектор-функции равен :

и матрица Гессиан для будет выглядеть следующим образом:

Тогда градиент и Гессиан, записанные в матричной форме, задаются формулой:

и

Если точка является локальным минимумом для дважды непрерывно дифференцируемой функцией , то точка стационарной точкой, т.е. градиент (14) равен 0. И наоборот, достаточным условием для того, чтобы стационарная точка была локальным минимум, является положительно определенный Гессиан.

Метод Ньютона основан на минимизации второго порядка рядом Тейлора

Минимизация получается путем приравнивания к нулю производной по *s*

Из (14) и (15) получаем итерационный процесс Ньютона:

Пренебрегая в (18) и заменяя , получим итерационный метод Ньютона-Гаусса

# ПРАКТИЧЕСКАЯ ЧАСТЬ

Показания всех зондов прибора электромагнитного каротажа являются функциями трёх переменных – логарифмов удельного электрического сопротивления породы и расстояния до границы пластов (). Зондом традиционно называют комбинацию используемых для измерения катушек (генерирующей поле и приёмной), частоты излучения и способа замера (изменение амплитуд или фаз). Для наших целей достаточно считать эти показания просто некоторыми числами, зависящими от x,y,z. Задача детерминированной инверсии заключается в определении параметров среды (по показаниям приборов.

Рисунок 1 – схема зонда

Модельные (или экспериментально полученные) показания приборов являются нам заданы и являются функциями, представленными в виде RBF-интерполяции

с узловыми точками и весами .В нашем случае ядро радиально-базисной функции имеет вид с известным .

Тестовые значения содержатся в файле rbf.txt, предоставленным научным руководителем.

# ЗАКЛЮЧЕНИЕ

В рамках курсовой работы были изучены и реализованы методы оптимизации, а именно методы Ньютона-Гаусса и Нелдера-Мида. Для каждого метода были реализованы программы на языке программирования C++, которые позволяли провести сравнение методов на сходимость и время выполнения.

# СПИСОК ЛИТЕРАТУРЫ

1. Обратные задачи и методы их решения. Приложения к геофизике : учебное пособие / А. Г. Ягола, Я. Ван, И. Э. Степанова, В. Н. Титаренко ; художник Н. А. Новак. — 4-е изд. — Москва : Лаборатория знаний, 2021. — 219 с.
2. Численные методы. Н.Н. Калиткин. Главная редакция физико-математическая литературы изд-ва «Наука», М., 1978
3. Волков, В. М. Численные методы : учебно-методическое пособие : в 2 частях / В. М. Волков. — Минск : БГУ, 2016 — Часть 1 — 2016. — 87 с.
4. Вагин, Д. В. Оценивание параметров в обратных задачах : учебное пособие / Д. В. Вагин. — Новосибирск : НГТУ, 2019. — 48 с.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.8)

project(kr

VERSION 0.0.2

LANGUAGES CXX

)

add\_subdirectory(src)

add\_subdirectory(example)

example/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

add\_executable(example main.cpp)

configure\_file(${CMAKE\_CURRENT\_SOURCE\_DIR}/../data/rbf.txt ${CMAKE\_CURRENT\_BINARY\_DIR}/rbf.txt)

set\_target\_properties(

example PROPERTIES

CXX\_STANDARD 17

CXX\_STANDARD\_REQUIRED ON

)

target\_include\_directories(

example

PRIVATE

${CMAKE\_CURRENT\_SOURCE\_DIR}/../include/methods\_lib

)

target\_include\_directories(

example

PRIVATE

${CMAKE\_CURRENT\_SOURCE\_DIR}/../include/Eigen

)

target\_link\_libraries(example optimization\_methods)

**example/main.cpp**

#include "windows.h"

#include "../include/methods\_lib/Methods.h"

int main() {

SetConsoleOutputCP(CP\_UTF8);

RBF rbf = RBF::ReadFile("rbf.txt");

VectorXd param(3), y;

param << 10, 100, 1;

rbf.getValues(param, y);

cout << "\nНелдера Мида\n";

VectorXd x(3);

x << 10, 12, 1.2;

Function function1(x);

function1.set(rbf, y);

method\_Neldera\_and\_Mida(function1);

function1.getX();

function1.getCountIter();

function1.getTime();

cout << "\nНьютона Гаусса\n";

Function function2(x);

function2.set(rbf, y);

method\_Newton\_and\_Gauss(function2);

function2.getX();

function2.getCountIter();

function2.getTime();

cout << "\nГибрид\n";

Function function3(x);

function3.set(rbf, y);

function3.hybrid\_mode();

function3.getX();

function3.getCountIter();

function3.getTime();

return 0;

}

**include/data/rbfreader.h**

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <cmath>

#include "../Eigen/Dense"

using namespace Eigen;

class RBF {

private:

size\_t n = 0, m = 0;

double e = 0, e2 = 0;

std::vector<std::vector<double> > data;

public:

size\_t getN() { return n; }

size\_t getM() { return m; }

static RBF ReadFile(std::string fileName) {

RBF r;

std::ifstream f(fileName);

if (!f.is\_open()) {

std::cout << "File not found!\n";

exit(1);

}

std::string s;

f >> r.n >> r.m >> r.e >> s;

r.e2 = r.e \* r.e;

double x, y, z;

std::vector<double> v(3 + r.m);

for (size\_t i = 0; i < r.n; i++) {

for (size\_t k = 0; k < 3 + r.m; k++) {

f >> v[k];

}

r.data.push\_back(v);

}

return r;

}

// p - номер датчика , x y z - три параметра (x=log10 (rho1),y=log10(rho2), z=h - расстояние до границы)

void getValues(VectorXd parametr, VectorXd &result) {

int x = parametr[0], y = parametr[1], z = parametr[2];

result.resize(m);

result.setZero();

for (size\_t k = 0; k < n; k++) {

double r2 = (x - data[k][0]) \* (x - data[k][0]) +

(y - data[k][1]) \* (y - data[k][1]) +

(z - data[k][2]) \* (z - data[k][2]);

double coeff = sqrt(r2 / e2 + 1);

for (size\_t p = 0; p < m; p++) {

result[p] += data[k][3 + p] \* coeff;

}

}

}

RBF() {}

};

**include/methods\_lib/Function.h**

#pragma once

#include "../Eigen/Dense"

#include <iostream>

#include <fstream>

#include <vector>

#include <random>

#include <ctime>

#include <functional>

#include <iomanip>

#include <chrono>

#include "../data/rbfreader.h"

using namespace Eigen;

using namespace std;

class Function {

public:

RBF rbf;

double \_time = 0.0;

int count\_iter = 0;

const double epsilon = 1e-2;

VectorXd x0, y, res;

int size = 0;

int count\_step = 0;

bool on\_hybrid = 0;

Function(VectorXd &x);

void set(RBF &rbf, VectorXd &y);

double mnk(VectorXd &x);

void print\_result();

void write\_data\_file(char \*name);

int getCountIter() const;

double getTime() const;

const VectorXd &getX() const;

void hybrid\_mode(int count\_iter);

~Function();

};

**include/methods\_lib/Methods.h**

#pragma once

#include "Method\_MelderaNida.h"

#include "Method\_NewtonAndGauss.h"

#include "../data/rbfreader.h"  
include/methods\_lib/ Method\_NewtonAndGauss.h  
#pragma once

**#include "Function.h"**

using namespace std;

using namespace Eigen;

const int max\_iter = 2000;

const double \_dx = 0.5;

void method\_Newton\_and\_Gauss(Function &function);

MatrixXd Jacobian(Function &function);

VectorXd calculation\_r(Function &function);

include/methods\_lib/ Method\_MelderaNida.h

#pragma once

#include "Function.h"

const double len\_a = 0.5;

const double alfa = 1;

const double gamma = 2;

const double betta = 0.5;

void method\_Neldera\_and\_Mida(Function &function);

MatrixXd set\_node\_fill(Function &function);

void set\_node\_fill\_coef(MatrixXd &node,Function &function);

VectorXd get\_central\_gravied(MatrixXd &node, int n, Function &function);

MatrixXd sort\_node(MatrixXd &node, int n);

VectorXd display(VectorXd &x\_central, VectorXd &x\_big, Function &function);

VectorXd stretching(VectorXd &x\_central, VectorXd &x\_display, Function &function);

VectorXd сompression(VectorXd &x\_central, VectorXd &x\_big, Function &function);

MatrixXd reduction(MatrixXd &node, int n, Function &function);

bool stopping(VectorXd &x, int n, Function &function);

**src/CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.8)

file(GLOB SRCS \*.cpp)

add\_library(

optimization\_methods

STATIC

${SRCS}

)

target\_include\_directories(

optimization\_methods

PRIVATE

${CMAKE\_CURRENT\_SOURCE\_DIR}/../include/Eigen)

target\_include\_directories(

optimization\_methods

PRIVATE

${CMAKE\_CURRENT\_SOURCE\_DIR}/../include/methods\_lib

)

set\_target\_properties(

optimization\_methods PROPERTIES

CXX\_STANDARD 17

CXX\_STANDARD\_REQUIRED ON

)

**src/Fuction.cpp**

#include "../include/methods\_lib/Function.h"

#include "Method\_MelderaNida.h"

#include "Method\_NewtonAndGauss.h"

void Function::write\_data\_file(char \*name) {

std::ofstream data\_file;

data\_file.open(name, std::ios::out);

data\_file.close();

}

Function::Function(VectorXd &x) {

this->x0 = x;

res.resize(x.size());

res = x;

}

Function::~Function() {

x0.resize(0);

y.resize(0);

}

double Function::mnk(VectorXd &x) {

double sum = 0;

VectorXd current\_res(y.size());

rbf.getValues(x, current\_res);

for (int i = 0; i < y.size(); i++) {

sum += (y[i] - current\_res[i]) \* (y[i] - current\_res[i]);

}

return sum;

}

void Function::print\_result() {

cout << "Начальные данные:\n";

cout << x0.transpose() << endl;

cout << "Параметры x\n";

cout << res.transpose() << endl;

}

void Function::set(RBF &rbf, VectorXd &y) {

this->rbf = rbf;

size = y.size();

this->y.resize(size);

this->y = y;

}

int Function::getCountIter() const {

cout << "count\_iter: " << count\_iter << endl;

return count\_iter;

}

const VectorXd &Function::getX() const {

cout << "Parametr: " << res.transpose() << endl;

return res;

}

double Function::getTime() const {

cout << setprecision(10) << "time: " << \_time << endl;

return \_time;

}

void Function::hybrid\_mode(int count\_iter) {

this->count\_step = count\_iter;

this->on\_hybrid = true;

method\_Neldera\_and\_Mida(\*this);

method\_Newton\_and\_Gauss(\*this);

}  
**src/ Method\_MelderaNida.cpp**  
#include "../include/methods\_lib/Methods.h"

MatrixXd set\_node\_fill(Function &function) {

int n = function.x0.size();

MatrixXd node(n, n);

node.setZero();

node.row(0) = function.x0.transpose();

set\_node\_fill\_coef(node, function);

return node;

}

void set\_node\_fill\_coef(MatrixXd &node, Function &function) {

int n = function.x0.size() - 1;

double a = ((sqrt(n + 1) - 1) / (n \* sqrt(2))) \* len\_a;

double b = ((sqrt(n + 1) + n - 1) / (n \* sqrt(2))) \* len\_a;

// cout << "a=" << a << " b=" << b << endl;

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

VectorXd x\_cur(n + 1);

x\_cur.setZero();

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

if (i == j) {

x\_cur(j) = function.x0(j) + a;

} else {

x\_cur(j) = function.x0(j) + b;

}

}

x\_cur(n) = function.mnk(x\_cur);

node.row(i + 1) = x\_cur.transpose();

}

}

void print\_node(const MatrixXd &node\_x);

void print\_point(VectorXd &x\_central);

void method\_Neldera\_and\_Mida(Function &function) {

clock\_t start = clock();

function.x0.conservativeResize(function.x0.size() + 1);

int n = function.x0.size() - 1;

MatrixXd node\_x = set\_node\_fill(function);

VectorXd funk = node\_x.col(n);

while (stopping(funk, n, function)) {

node\_x = sort\_node(node\_x, n);

VectorXd x\_central = get\_central\_gravied(node\_x, n, function);

VectorXd x\_big = node\_x.row(n);

VectorXd x\_new = display(x\_central, x\_big, function);

if (x\_new(n) < x\_big(n)) {

node\_x.row(n) = x\_new.transpose();

VectorXd x\_stretching = stretching(x\_central, x\_new, function);

if (x\_stretching(n) < x\_new(n)) {

node\_x.row(n) = x\_stretching.transpose();

node\_x = sort\_node(node\_x, n);

} else {

node\_x = sort\_node(node\_x, n);

}

} else {

if (x\_new(n) < node\_x(n - 1, n)) {

node\_x.row(n) = x\_new.transpose();

node\_x = sort\_node(node\_x, n);

} else {

VectorXd x\_compression = сompression(x\_central, x\_big, function);

if (x\_compression(n) < x\_big(n)) {

node\_x.row(n) = x\_compression.transpose();

node\_x = sort\_node(node\_x, n);

} else {

node\_x = reduction(node\_x, n, function);

}

}

}

funk = node\_x.col(n);

function.count\_iter++;

if (function.count\_iter>function.count\_step && function.on\_hybrid){

break;

}

}

node\_x = sort\_node(node\_x, n);

funk = node\_x.row(0);

funk.conservativeResize(funk.size() - 1);

function.res=funk;

clock\_t end = clock();

function.\_time += ((double)(end - start) / CLOCKS\_PER\_SEC);

}

void print\_point(VectorXd &x\_central) { cout << "Точка:\n" << x\_central.transpose() << endl; }

void print\_node(const MatrixXd &node\_x) { cout << "Матрица:\n" << node\_x << endl; }

VectorXd get\_central\_gravied(MatrixXd &node, int n, Function &function) {

VectorXd cent\_grav(n + 1);

cent\_grav.setZero();

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

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

cent\_grav(i) += node(j, i);

}

cent\_grav(i) = cent\_grav(i) / n;

}

cent\_grav(n) = function.mnk(cent\_grav);

return cent\_grav;

}

MatrixXd sort\_node(MatrixXd &node, int n) {

int sortingRow = n;

VectorXi indices(node.rows());

for (int i = 0; i < node.rows(); ++i) {

indices(i) = i;

}

sort(indices.data(), indices.data() + indices.size(),

[&node, sortingRow](int i, int j) {

return node(i, sortingRow) < node(j, sortingRow);

});

MatrixXd sortedMatrix(node.rows(), node.cols());

for (int i = 0; i < node.rows(); ++i) {

sortedMatrix.row(i) = node.row(indices(i));

}

return sortedMatrix;

}

VectorXd display(VectorXd &x\_central, VectorXd &x\_big, Function &function) {

VectorXd new\_point = x\_central + alfa \* (x\_central - x\_big);

new\_point(x\_central.size() - 1) = function.mnk(new\_point);

return new\_point;

}

VectorXd stretching(VectorXd &x\_central, VectorXd &x\_display, Function &function) {

VectorXd new\_point = x\_central + gamma \* (x\_display - x\_central);

new\_point(x\_central.size() - 1) = function.mnk(new\_point);

return new\_point;

}

VectorXd сompression(VectorXd &x\_central, VectorXd &x\_big, Function &function) {

VectorXd new\_point = x\_central + betta \* (x\_big - x\_central);

new\_point(x\_central.size() - 1) = function.mnk(new\_point);

return new\_point;

}

MatrixXd reduction(MatrixXd &node, int n, Function &function) {

MatrixXd new\_node(n + 1, n + 1);

new\_node.setZero();

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

new\_node.row(i) = node.row(i) + 0.5 \* (node.row(i) - node.row(0));

VectorXd new\_point = new\_node.row(i);

new\_node(i, n) = function.mnk(new\_point);

}

return new\_node;

}

bool stopping(VectorXd &x, int n, Function &function) {

double value\_foo = x.sum() / (n + 1);

double sigma = 0;

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

sigma = (x(i) - value\_foo);

}

sigma = sqrt(sigma / (n + 1));

if (sigma < function.epsilon) {

return 0;

} else {

return 1;

}

}

**src/ Method\_NewtonAndGauss.cpp**  
#include "../include/methods\_lib/Methods.h"

void method\_Newton\_and\_Gauss(Function &function) {

clock\_t start = clock();

int count = function.x0.size();

VectorXd old(count);

for (int i = 0; i < max\_iter; i++) {

old = function.res;

MatrixXd J = Jacobian(function);

VectorXd dy = calculation\_r(function);

function.res = old - (J.transpose() \* J).inverse() \* J.transpose() \* dy;

if ((old - function.res).norm() < function.epsilon) {

function.count\_iter += i;

break;

}

}

clock\_t end = clock();

function.\_time += (double) (end - start) / CLOCKS\_PER\_SEC;

}

MatrixXd Jacobian(Function &function) {

int row = function.size, col = function.res.size();

MatrixXd Jacobian(row, col);

Jacobian.setZero();

VectorXd dx(col);

VectorXd grad\_l(col), grad\_r(col);

VectorXd y\_l, y\_r;

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

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

dx.setZero();

dx[j] = \_dx;

grad\_l = function.res + dx;

grad\_r = function.res - dx;

function.rbf.getValues(grad\_l, y\_l);

function.rbf.getValues(grad\_r, y\_r);

Jacobian(i, j) = (-y\_l[i] + y\_r[i]) / (2 \* \_dx);

}

}

return Jacobian;

}

VectorXd calculation\_r(Function &function) {

int col = function.size;

VectorXd r(col);

VectorXd current\_res;

function.rbf.getValues(function.res, current\_res);

r = function.y - current\_res;

return r;

}

**ПЛАН-ГРАФИК**

**выполнения курсовой работы**

обучающегося Акмурзин М.Э

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа работ | Трудоемкость выполнения, час. | Процент к общей трудоемкости выполнения | Срок предъявления консультанту |
| Получение и согласование задания | 0,3 | 0,8 | 6 неделя |
| Знакомство с литературой по теме курсовой работы | 2,7 | 7,5 | 7 неделя |
| Реализация метода Нелдера-Мида | 10 | 27,7 | 10 неделя |
| Реализация метода Ньютона Гаусса | 10 | 27,7 | 12 неделя |
| Сравнения методов | 10 | 27,7 | 14 неделя |
| Составление и оформление пояснительной записки и подготовка к защите | 2,7 | 7,5 | 16 неделя |
| Защита | 0,3 | 0,8 | 17 неделя |
| Итого | 36 | 100 | - |