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										

ЧИСЛЕННОЕ РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ С РАЗРЕЖЕННОЙ МАТРИЦЕЙ КОЭФФИЦИЕНТОВ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

«Численные методы»

3952.335213.000 ПЗ

Группа МКН-316	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Яковлев О.В.			
Консультант	Маякова С.А.			
Принял	Лукащук В.О.			

Уфа 2022

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: Яковлев Олег Витальевич Группа: МКН-316

Консультант: Маякова Светлана Алексеевна

1. Тема курсовой работы

Численное решение систем линейных уравнений с разреженной матрицей коэффициентов.

2. Основное содержание

2.1. Изучить литературу по теме "Численное решение систем линейных уравнений с разреженной матрицей коэффициентов"
2.2. Разработать программу для решения СЛАУ с разреженной матрицей с помощью прямого и итерационного методов.
2.3. Сравненить зависимость производительности программы при решении СЛАУ от размера матрицы для каждого из методов.
2.4. Оформить пояснительную записку к курсовой работе.

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	Дата окончания работы
"__" ________ 202_ г.	"__" __________ 202_ г.

Консультант __________________ Маякова С.А.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Часто в процессе решения дифференциальных уравнений возникает задача решения системы линейных алгебраических уравнений с симметричной матрицей обладающей диагональным преобладанием. Развитие вычислительной техники и вызванный этим процессом переход к более сложным (трехмерным, в произвольных геометрических областях) моделям в виде систем дифференциальных уравнений в частных производных, привел к необходимости решения больших разреженных систем линейных алгебраических уравнений.

Целью данной курсовой работы является изучение способов решения СЛАУ с разреженной матрицей. Для достижения данной цели были поставлены следующие задачи:

1. Изучить литературу по теме "Численное решение систем линейных уравнений с разреженной матрицей коэффициентов"

2. Разработать программу для решения СЛАУ с разреженной матрицей с помощью прямых и итерационных методов.

3. Сравнить производительности программы при решении СЛАУ для каждого из методов.

1. ХРАНЕНИЕ И ОБРАБОТКА РАЗРЕЖЕННЫХ МАТРИЦ

Распространенным способом хранения несимметричных разряженных матриц произвольной структуры является CSR. В нем разреженная матрица A хранится с использованием следующих массивов:

- `value`, который содержит значения всех ненулевых элементов матрицы.
- `cols`, который содержит номера столбцов, в которых стоит каждый ненулевой элемент.
- `rows`, который содержит индекс начала строки в массивах `values` и `cols`.

Умножение матрицы, хранимой в таком формате, реализуется с помощью перебора элементов массива `values`, которые умножаются на элементы вектора x с индексом из соответствующего элемента массива `cols`.[2]

2. РЕШЕНИЕ СЛАУ ПРИ ПОМОЩИ МЕТОДА LU-РАЗЛОЖЕНИЯ

LU-разложение – это представление матрицы A в виде произведения двух матриц, $A = LU$, где $L = (l_{ij})$ – нижняя треугольная матрица, а $U = (u_{ij})$ – верхняя треугольная матрица с единичной диагональю. Элементы l_{ij}, $u_{i,j}$ определим из условия

$$\sum_{k=1}^{n} l_{ik} u_{kj} = a_{ij} \quad \forall i, j = 1, \dots, n$$

где матрицы L и U имеют вид

$$L = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{pmatrix}, U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

Перемножив матрицы, получаем формулы для элементов матриц L и U.

$$\begin{aligned}
&l_{i1} = a_{i1} \\
&u_{1i} = \frac{a_{1j}}{l_{11}} \\
&l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} && i \geq j > 1 \\
&u_{ij} = \frac{1}{u_{ii}}\left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right) && 1 < i < j
\end{aligned}$$

Далее, СЛАУ можно представить в виде $LUx = b$. Решить систему можно разделив ее на две системы, имеющие треугольные матрицы.

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Эти системы можно решить с помощью процедур прямого и обратного хода.[3]

3. МЕТОД БИСОПРЯЖЕННЫХ ГРАДИЕНТОВ

Системы векторов $\{x\}_{i=1}^{m}$ и $\{y\}_{i=1}^{m}$ называются биортогональными, если скалярное произведение (x_i, y_i) обращается в ноль при $i \neq j$.

Пусть векторы v_1 и w_1 таковы, что $(v1, w1) \neq 0$ и пусть системы векторов $\{y\}_{i=1}^{m}$ и $\{w\}_{i=1}^{m}$ определяются соотношениями:

$$\begin{aligned}
v_{i+1} &= Av_i - \alpha_i v_i - \beta_i v_{i-1} && v_0 = 0 \\
w_{i+1} &= A^T w_i - \alpha_i w_i - \beta_i w_{i-1} && w_0 = 0 \\
\alpha_i &= \frac{(Av_i, w_i)}{(v_i, w_i)} \\
\beta_i &= \frac{(v_i, w_i)}{(v_{i-1}, w_{i-1})} && \beta_1 = 0
\end{aligned}$$

тогда системы $\{y\}_{i=1}^{m}$ и $\{w\}_{i=1}^{m}$ биортогональны и каждая из них линейно независима и образует базис в $K_m(v_1, A)$ и $K_m(w_1, A^T)$ соответственно.

Аналогично методу полной ортогонализации решение системы будет уточняться по формуле

$$x_m = x_0 + \beta V_m T_m^{-1} e_1$$

запишем LU-разложение для матрицы T_m

$$T_m = L_m U_m$$

Пусть $P_m = V_m U_m^{-1}$. Тогда x_m имеет вид

$$x_m = x_0 + \beta P_m L_m^{-1} e_1$$

Аналогично определим $\bar{P}_m$

$$\bar{P}_m = W_m (L_m^T)^{-1}$$

Тогда имеет место

$$\bar{P}_m^T A P_m = I_m D_m$$

где $D_m = (d_{ii}) = (v_i, w_i)$.

Таким образом, можно составить алгоритм 1 решения СЛАУ с помощью метода бисопряженных градиентов. [1]

Algorithm 1 Метод бисопряженных градиентов

Выбрать начальное приближение x_0
$r_0 \leftarrow b - Ax_0$
Выбрать вектор $\tilde{r}_0$, такой что $(r_0, \tilde{r}_0) \neq 0$
$p_0 \leftarrow r_0$
$\tilde{p}_0 \leftarrow \tilde{r}_0$
Цикл j=1,2,. . . **выполнять**
$\quad \alpha_j \leftarrow \frac{(r_j, \tilde{r}_j)}{(Ap_j, \tilde{p}_j)}$
$\quad x_{j+1} \leftarrow x_j + \alpha_j p_j$
$\quad r_{j+1} \leftarrow r_j - \alpha_j A p_j$
$\quad \tilde{r}_{j+1} \leftarrow \tilde{r}_j - \alpha_j A^T \tilde{p}_j$
$\quad \beta_j \leftarrow \frac{(r_{j+1}, \tilde{r}_{j+1})}{(r_j, \tilde{r}_j)}$
$\quad p_{j+1} \leftarrow r_{j+1} + \beta_j p_j$
$\quad \tilde{p}_{j+1} \leftarrow \tilde{r}_{j+1} + \beta_j \tilde{p}_j$
Конец цикла

При решении СЛАУ с помощью итерационных методов, необходимо также знать достаточные условия сходимости данных методов. Одним из таких условий может быть диагональное преобладание матрицы системы.

4. ГЕНЕРАЦИЯ СИММЕТРИЧНОЙ МАТРИЦЫ С ДИАГОНАЛЬНЫМ ПРЕОБЛАДАНИЕМ

Сперва для элементов матрицы, не лежащих на главной диагонали, генерируются случайные действительные числа из интервала $[-1; 1]$. Далее считается сумма модулей этих элементов.

$$a_{ii} = \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|$$

Значение этой суммы является значением элемента на главной диагонали данной строки матрицы.

Далее необходимо сделать матрицу симметричной. Для этого элементам, расположенным ниже главной диагонали присваиваем те же значения, что и у соответствующих элементов выше главной диагонали.

$$a_{ij} := a_{ji} \quad j < i$$

5. РЕЗУЛЬТАТЫ

5.1. Разработка структуры классов программной реализации

В ходе выполнения курсовой работы был разработан класс для работы с разреженными матрицами `compressed_matrix`. Класс реализует описанный ранее способ хранения разреженной матрицы CSR и методы для работы с разреженными матрицами:

- `LU_decomposition` – LU-разложение матрицы
- `solve_L` – прямой ход метода Гаусса для решения СЛАУ с нижней треугольной матрицей
- `solve_U` – обратный ход метода Гаусса для решения СЛАУ с верхней треугольной матрицей
- `solve_BiCGStab` – решение СЛАУ с помощью метода бисопряженных градиентов
- `check` – проверяет, что вектор решения удовлетворяет СЛАУ.

5.2. Анализ производительности программной реализации.

При решении СЛАУ малой размерности ($N \approx 10$), прямые могут оказаться быстрее. Однако, при решении СЛАУ с матрицей больших размерностей, эффективны итерационные методы.

На рисунке 1 изображен график зависимости времени решения СЛАУ с разреженной матрицей (матрица заполнена на 20%, $\varepsilon = 10^{-5}$) при помощи прямого метода LU-разложения матрицы системы и при помощи итерационного стабилизированного метода бисопряженных градиентов(BiCGStab). График показывает, что скорость сходимости итерационного метода сильно зависит от коэффициентов матрицы системы. Поэтому решение систем с матрицей, обладающих сильным диагональным преобладанием происходит приблизительно в 2 раза быстрее. Время решения СЛАУ с помощью метода LU-разложения не зависит от диагонального преобладания матрицы системы.

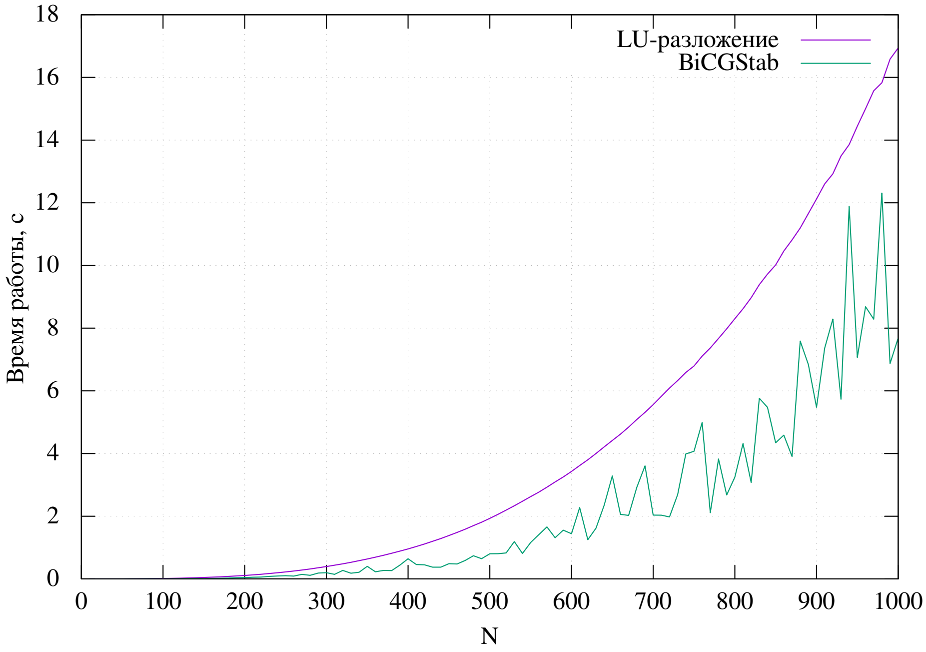

Рис. 1: Зависимость времени выполнения программы от размера матрицы.

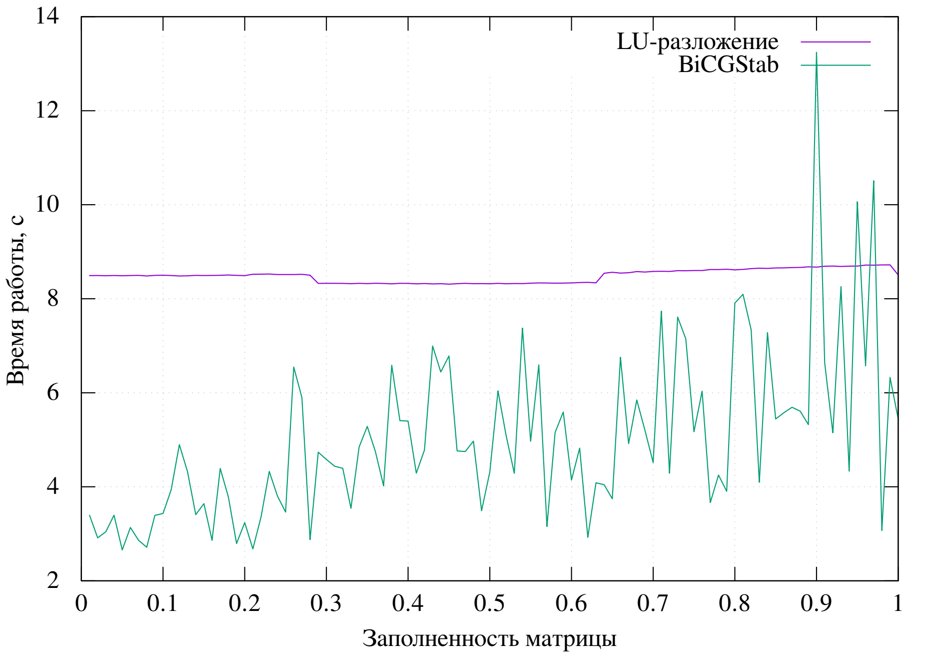

Рис. 2: Зависимость времени выполнения программы от заполненности матрицы.

Также время решения системы может зависеть от разреженности матрицы. На рисунке 2 изображен график зависимости времени решения СЛАУ($N = 800$, $\varepsilon = 10^{-5}$) методами LU-разложения и BiCGStab от заполненности матрицы(отношения количества ненулевых коэффициентов матрицы к общему их количеству). На графике видно, что при увеличении заполненности время решения СЛАУ с помощью итерационного метода тоже увеличивается. Поэтому для систем небольшой размерности с плотно заполненной матрицей можно применять прямые методы решения СЛАУ.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была изучена литература по теме «Численное решение систем линейных уравнений с разреженной матрицей коэффициентов». Далее была разработана программа для решения СЛАУ с разреженной матрицей с помощью прямых и итерационных методов. Наконец было произведено сравнение производительности программы при решении СЛАУ для каждого из методов.

В ходе сравнения было показано, что при решении СЛАУ большой размерности в 2 раза большую производительность показывают итерационные методы по сравнению с прямыми методами. Время решения СЛАУ с помощью метода LU-разложения не зависит от диагонального преобладания матрицы системы. При увеличении заполненности время решения СЛАУ с помощью итерационного метода тоже увеличивается.

Таким образом были изучены способы решения СЛАУ с разреженной матрицей.

СПИСОК ЛИТЕРАТУРЫ

1. *Saad Y.* Iterative Methods for Sparse Linear Systems. — Second. — SIAM, 2003. — ISBN 978-0-89871-534-7. — URL: `https://www-users.cse.umn.edu/~saad/IterMethBook_2ndEd.pdf`.
2. *Баландин М. Ю.*, *Шурина Э. П.* Методы решения СЛАУ большой размерности. — Новосибирск : Изд-во НГТУ, 2000. — 70 с.
3. *Калиткин Н. Н.* Численные методы. — М. : Издательство «Наука», 1978.

ПРИЛОЖЕНИЕ А

Листинг программы

main.cpp

```
#include <fstream>
#include <iostream>
#include "compressed_matrix.hpp"
#include <ctime>



int main(){
    compressed_matrix m(5);
    compressed_matrix L(5);
    compressed_matrix U(5);
    //for(int i=0;i<100;i++){
        int N = 1000;


        std::vector<double> x(N,0),y(N),b(N);
        /*
        m.read("mat.txt");
        std::cout<<"M = "<<std::endl;
        m.print_matrix();
        std::cout<<std::endl;
        for(int i=0;i<5;i++){
            for(int j=0;j<5;j++){
                std::cout<<m.get(i,j)<<' ';
            }
            std::cout<<std::endl;
        }
        m.ILU_decomposition(L, U);
        std::cout<<"L:\n";
        L.print_matrix();
        std::cout<<std::endl;
        std::cout<<"U:\n";
        U.print_matrix();
        std::cout<<std::endl;
        */
        double q = 0.20;
        //std::cout<<q<<' ';

        m = compressed_matrix(N);
        b = std::vector<double>(N,1);
        m.generate(1338,q);
        L = compressed_matrix(N);
        U = compressed_matrix(N);
        auto lu_start = clock();
        m.LU_decomposition(L,U);
        L.solve_L(b,y);
        U.solve_U(y,x);
        auto lu_end = clock();
        std::cout<<"LU time:"<<(lu_end -
           lu_start*1.0)/CLOCKS_PER_SEC<<std::endl;
        if(m.check(b,x))
            std::cout<<"[OK]"<<std::endl;
        else
            std::cout<<"[FAIL]"<<std::endl;
        x = std::vector<double>(N,0);
        auto bcg_start = clock();
        double r = m.BiCGStab_solve(b,x,10000);
        auto bcg_end = clock();
```

```
        std::cout<<"BiCGStab␣time:"<<(bcg_end -
            bcg_start*1.0)/CLOCKS_PER_SEC<<std::endl;
        std::cout<<"Residue␣norm:"<<r<<std::endl;
        if(m.check(b,x))
            std::cout<<"[OK]"<<std::endl;
        else
            std::cout<<"[FAIL]"<<std::endl;
    //}

}
```

compressed_matrix.hpp

```
#pragma once
#include <vector>
#include <string>
class compressed_matrix{
    protected:
        std::vector<int> rows;
        std::vector<int> cols;
        std::vector<double> value;
        void set_init(int i,int j, double v);
                void set_size(int n);
        virtual double mul_sub(const compressed_matrix& other,int i,int
            j,int k);
    public:
        compressed_matrix(int n);
                void read( const std::string & filename);
        void generate(int seed,double chance=0.5);
        virtual ~compressed_matrix();
        int row_num() const;
        int elem_num() const;
        void print_matrix();
        double get(int i,int j) const ;
                void LU_decomposition(compressed_matrix
                    &L,compressed_matrix &U);
        void ILU_decomposition(compressed_matrix &L,compressed_matrix &U);
                void solve_L(const std::vector<double> &b,
                    std::vector<double>& y);
                void solve_U(const std::vector<double> &y,
                    std::vector<double> &x);
        std::vector<double> operator*(const std::vector<double>& vec)const ;

        std::vector<double> T_prod(const std::vector<double>& vec)const ;
                double BiCGStab_solve(const std::vector<double> &b,
                    std::vector<double>& x,int n = 1000);
        bool check(std::vector<double> b, std::vector<double> x);
};
```

compressed_matrix.cpp

```
#include "compressed_matrix.hpp"
#include <limits.h>
#include <utility>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdlib>
compressed_matrix::compressed_matrix(int n){
    set_size(n);
}
void compressed_matrix::set_size(int n){
        rows.clear();
        rows.resize(n+1,n*n);
```

```
    rows[0] = 0;
        cols.clear();
        value.clear();
}
int compressed_matrix::elem_num()const{
    return value.size();
}
int compressed_matrix::row_num()const{
    return rows.size()-1;
}

void compressed_matrix::print_matrix(){
    std::cout<<"[";
        for(int i = 0;i<row_num();i++){
                auto begin = rows[i];
                auto end = rows[i+1];
        std::cout<<"[";
                for(int j = 0; j< row_num();j++){
                        double a = 0;
                        if(begin<end&&begin<row_num()*row_num()&&j ==
                            cols[begin]){
                                a = value[begin];
                                begin++;
                        }


                        std::cout<<a<<", ";
                }
                std::cout<<"],"<<std::endl;
        }
    std::cout<<"]";
}
void compressed_matrix::set_init(int i, int j, double v){
    rows[i] = std::min(rows[i],(int)value.size());
        value.push_back(v);
        cols.push_back(j);
}
void compressed_matrix::read(const std::string &filename){
        using namespace std;
        ifstream mat_file(filename);
        int n,k;
        mat_file>> k>> n;
        set_size(k);
        for(int i=0;i<n;i++){
                int ii,jj,v;
                mat_file >> ii>>jj>>v;
                set_init(ii,jj,v);
        }
}
double compressed_matrix::get(int i,int j) const{
    if(rows[i]>=cols.size())
        return 0.0;
    auto it =
        std::lower_bound(cols.begin()+rows[i],cols.begin()+std::min(rows[i+1],(int)cols.
    if(it!=cols.end()&&*it == j){
        return value[it - cols.begin()];
    }
    return 0.0;
}
double compressed_matrix::mul_sub(const compressed_matrix& other,int i,int
    j,int k){
    double sum = 0.0;
    /*
    for(int ri=rows[i];ri<rows[i+1];ri++){
        std::cout<<"ri = "<<ri<<std::endl;
```

```
        if(ri>=cols.size())
            continue;
        int rk = cols[ri];
        std::cout<<"rk = "<<rk<<std::endl;
        if(rk>=k)
            break;
        double rv = value[rk];
        double ov = other.get(rk,j);
        sum+=rv*ov;
        std::cout<<"("<<ri<<","<<rk<<") "<<rv<<" * "<<ov<<std::endl;


    }
    */
    for(int t=0;t<k;t++){
        auto a = get(i,t);
        auto b = other.get(t,j);
        //std::cout<<"("<<a<<"*"<<b<<":"<<t<<") " ;
        sum+=a*b;
    };
    //std::cout<<std::endl;
    return sum;
}
void compressed_matrix::LU_decomposition(compressed_matrix &L,
    compressed_matrix &U){
    L.set_size(row_num());
    U.set_size(row_num());
    for(int i=0;i<row_num();i++){
        for(int j=0;j<row_num();j++){
            if(i<=j){
                //if(i==j)
                //    L.set_init(i,j,1);
                double t = (get(i,j)-L.mul_sub(U,i,j,i));
                //std::cout<< "U["<<i<<","<<j<<"]::"<<t<<std::endl;
                U.set_init(i,j,t);
                //U.print_matrix();
                if(i==j)
                    L.set_init(i,j,1);

            }else{
                //std::cout<<"M[i,j] = "<<get(i,j)<<std::endl;
                double t = (get(i,j)-L.mul_sub(U,i,j,j))/U.get(j,j);
                //std::cout<< "L["<<i<<","<<j<<"]::"<<t<<"/ U[j,j]
                    ="<<U.get(j,j)<<std::endl;
                L.set_init(i,j,t);
                //L.print_matrix();
            }
        }
    }


}
void compressed_matrix::ILU_decomposition(compressed_matrix &L,
    compressed_matrix &U){
    L.set_size(row_num());
    U.set_size(row_num());
    for(int i=0;i<row_num();i++){
        bool diag = false;
        for(int r=rows[i];r<std::min(rows[i+1],(int)cols.size());r++){
            int j = cols[r];
            double v= value[r];
            if(i<=j){
                //if(i==j)
                //    L.set_init(i,j,1);
                double t = (v-L.mul_sub(U,i,j,i));
```

```
                //std::cout<< "U["<<i<<","<<j<<"]::"<<t<<std::endl;
                U.set_init(i,j,t);
                //U.print_matrix();
                if(!diag){
                    L.set_init(i,i,1);
                    diag = true;
                }

            }else{

                //std::cout<<"M[i,j] = "<<get(i,j)<<std::endl;
                double t = (v-L.mul_sub(U,i,j,j))/U.get(j,j);
                //std::cout<< "L["<<i<<","<<j<<"]::"<<t<<"| U[j,j]
                    ="<<U.get(j,j)<<std::endl;
                L.set_init(i,j,t);
                //L.print_matrix();
            }
        }
        if(!diag){
            L.set_init(i,i,1);
            diag = true;
        }
    }
}
void compressed_matrix::solve_L(const std::vector<double> &b,
    std::vector<double> &y){
    y = std::vector<double>(b);
    for(int i=0;i<row_num();i++){
        y[i]/=get(i,i);
        for(int j=i+1;j<row_num();j++){


            y[j]-=get(j,i)*y[i];
        }
    }
}
void compressed_matrix::solve_U(const std::vector<double> &y,
    std::vector<double> &x){
    x = std::vector<double>(y);
    for(int i=row_num()-1;i>=0;i--){
        x[i]/=get(i,i);
        for(int j=i-1;j>=0;j--){
            x[j]-=get(j,i)*x[i];
        }
    }

}
std::vector<double> compressed_matrix::operator*(const std::vector<double>&
    vec)const {
    std::vector<double> res(row_num(),0);
    for(int i=0;i<res.size();i++){
        for(int j=rows[i];j<std::min(rows[i+1],(int)cols.size());j++){
            res[i]+= value[j] * vec[cols[j]];
        }
    }
    return res;

}
std::vector<double> operator+(const std::vector<double> &a, const
    std::vector<double> &b){
    std::vector<double> c(a.size());
    for(int i=0;i<a.size();i++){
        c[i]=a[i]+b[i];
    }
    return c;
```

```
}
std::vector<double> operator*(double a, const std::vector<double> &b){
    std::vector<double> res(b);
    for(double &bi : res)
        bi*=a;
    return res;
}
std::vector<double> operator-(const std::vector<double> &a, const
    std::vector<double> &b){
    std::vector<double> c(a.size());
    for(int i=0;i<a.size();i++){
        c[i]=a[i]-b[i];
    }
    return c;
}

double operator*(const std::vector<double> &a, const std::vector<double>
    &b){
    double sum=0.0;
    for(int i=0;i<a.size();i++){
        sum+=a[i]*b[i];
    }
    return sum;
}
std::vector<double> operator/(std::vector<double> &a, double b){
    std::vector<double> c(a.size());
    for(int i=0;i<a.size();i++){
        c[i]=a[i]/b;
    }
    return c;
}

double norm(const std::vector<double> &x){
    double sum = 0;
    for(double s:x)
        sum+=s*s;
    return sqrt(sum);
}

std::vector<double> compressed_matrix::T_prod(const std::vector<double>
    &vec)const {
    std::vector<double> res(vec.size(),0);
    for(int i=0;i<vec.size();i++){
        for(int k =0;k<row_num();k++){
            res[i]+=get(k,i)*vec[k];

        }

    }
    return res;
}

double compressed_matrix::BiCGStab_solve(const std::vector<double> &b,
    std::vector<double>& x,int m){
    const compressed_matrix &A = *this;
    std::vector<double> r0 = b - A*x;
    std::vector<double> r0_ = r0;
    std::vector<double> p0 = r0;
    std::vector<double> p0_ = r0;
    for(int j=0;j<m;j++){
        double alpha = (r0*r0_)/((A*p0) * r0_);
        std::vector<double> s = r0 - alpha * (A*p0);
        double omega = (A * s) * s/((A*s)*(A*s));
        x = x + alpha*p0 +omega * s;
        std::vector<double> r1 = s - omega * (A *s);
```

```
        double beta = (r1*r0_)/(r0*r0_)*alpha/omega;
        p0 = r1 + beta * (p0 - omega*(A*p0));
        r0 = r1;
        double rn = norm(r0);
        //std::cout<<"Err: "<<rn<<std::endl;
        if(rn<1e-5){
            return rn;
        }
        /*
        double alpha = (r0*r0_)/((A*p0) * p0_);
        x = x+ alpha*p0;
        auto r02 = r0 - alpha * (A * p0);
        auto r0_2 = r0_ - alpha * A.T_prod(p0_);
        double beta = (r02 *r0_2)/(r0*r0_);
        double rn = norm(r02);
        std::cout<<"Err: "<<rn<<std::endl;
        if(std::abs(rn)<1e-7){
            return;
        }
        if(std::abs(beta) < 1e-9)
            return;
        p0  = r02  + beta * p0;
        p0_ = r0_2 + beta * p0_;
        r0 = r02;
        r0_ = r0_2;*/
    }
    return norm(r0);
}
double rnd(){
    return rand()*2.0/RAND_MAX - 1.0;
}
void compressed_matrix::generate(int seed,double chance){
    srand(seed);
    for(int i=0;i<row_num();i++){
        double sum = fabs(rnd())*10;
        for(int j=0;j<row_num();j++){
            if(i==j)
                set_init(i,j,sum*2);
            else if(j<i){
                set_init(i,j,get(j,i));
            }else{
                if(sum>0&&fabs(rnd())<chance){
                    double val = sum*rnd();
                    sum-=fabs(val);
                    set_init(i,j,val);
                }
            }
        }
    }

}

bool compressed_matrix::check(std::vector<double> b, std::vector<double> x){
    std::vector<double> r = (*this)*x - b;
    for(int i=0;i<r.size();i++){
        if(std::abs(r[i])>=1e-5)
            return false;
    }
    return true;
}


compressed_matrix::~compressed_matrix(){

}
```

ПЛАН-ГРАФИК

выполнения курсовой работы

обучающегося Яковлева О.В.

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
Изучение способов хранения разреженных матриц	9	25	9 неделя
Изучение итерационных методов решения СЛАУ	9,5	26,389	11 неделя
Программная реализация методов решения СЛАУ с разреженной матрицей	10	27,778	13 неделя
Замер производительности и анализ результатов	7,5	20,833	15 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	-