Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Лабораторная работа №3
по дисциплине «Численные методы»

Решение разреженных СЛАУ трехшаговыми итерационными методами с предобусловливанием

Факультет:	ПМИ
Группа:	ПМ-63
Студент:	Шепрут И.И.
Вариант:	11
Преподаватель:	Задорожный А.Г.

Новосибирск
2018

1 Цель работы

Изучить особенности реализации трехшаговых итерационных методов для СЛАУ с разреженными матрицами. Исследовать влияние предобусловливания на сходимость изучаемых методов на нескольких матрицах большой (не менее 10000) размерности.

Вариант 11: Сравнить МСГ и ЛОС для несимметричной матрицы. Факторизация LU(sq).

2 Исследования

2.1 Матрица с диагональным преобладанием

$$A = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ -3 & 13 & -4 & 0 & 0 & -4 & 0 & -2 & 0 & 0 \\ 0 & 0 & 7 & -3 & 0 & 0 & -2 & 0 & -2 & 0 \\ 0 & 0 & -3 & 8 & -2 & 0 & 0 & 0 & 0 & -3 \\ -2 & 0 & 0 & -2 & 5 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 2 & 0 & 0 & 0 & 0 \\ -2 & 0 & -4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & 0 & 0 & -3 & 7 & -1 & 0 \\ 0 & 0 & -2 & 0 & -4 & 0 & 0 & -3 & 9 & 0 \\ 0 & 0 & 0 & -1 & 0 & -4 & 0 & 0 & -4 & 9 \end{pmatrix}, X = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}, F = \begin{pmatrix} -5 \\ -29 \\ -23 \\ -17 \\ 9 \\ 5 \\ 28 \\ 14 \\ 31 \\ 26 \end{pmatrix}$$

$$\varepsilon = 10^{-14}, \quad iterations_{max} = 10000, \quad start = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$$

Метод	Итераций	Относительная невязка	Время
Якоби	513	$9.3 \cdot 10^{-15}$	?
Гаусс-Зейдель	85	$9.1 \cdot 10^{-15}$	?
МСГ LU(sq)	10	$4.3 \cdot 10^{-16}$	11.26 мкс
ЛОС	10001	$2.2 \cdot 10^{-2}$	3.55 мс
ЛОС LU(sq)	36	$2.8 \cdot 10^{-15}$	32.7 мкс
ЛОС Диаг.	10001	$9.8 \cdot 10^{-3}$	3.95 мс

2.2 Матрица с обратным знаком внедиагональных элементов

$$B = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 13 & 4 & 0 & 0 & 4 & 0 & 2 & 0 & 0 \\ 0 & 0 & 7 & 3 & 0 & 0 & 2 & 0 & 2 & 0 \\ 0 & 0 & 3 & 8 & 2 & 0 & 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 2 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 3 & 7 & 1 & 0 \\ 0 & 0 & 2 & 0 & 4 & 0 & 0 & 3 & 9 & 0 \\ 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 4 & 9 \end{pmatrix}, X = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix}, F = \begin{pmatrix} 9 \\ 81 \\ 65 \\ 81 \\ 41 \\ 19 \\ 56 \\ 98 \\ 131 \\ 154 \end{pmatrix}$$

$$\varepsilon = 10^{-14}, \quad iterations_{max} = 10000, \quad start = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$$

Метод	Итераций	Относительная невязка	Время
Якоби	132	$7.7 \cdot 10^{-15}$	?
Гаусс-Зейдель	57	$9.9 \cdot 10^{-15}$	?
МСГ LU(sq)	10	$2.5 \cdot 10^{-19}$	18.66 мкс
ЛОС	119	$9.5 \cdot 10^{-15}$	38.26 мкс
ЛОС LU(sq)	26	$7.8 \cdot 10^{-15}$	16.22 мкс
ЛОС Диаг.	88	$6.6 \cdot 10^{-15}$	33.56 мкс

2.3 Большой тест 0945

Метод	Итераций	Относительная невязка	Время
МСГ	393	$9.7 \cdot 10^{-21}$	17.06 мс
МСГ LU(sq)	11	$5.4 \cdot 10^{-21}$	2.35 мс
МСГ Диаг.	50	$5.5 \cdot 10^{-21}$	2.41 мс
ЛОС	486	$9 \cdot 10^{-21}$	21.64 мс
ЛОС LU(sq)	9	$2.5 \cdot 10^{-21}$	1.36 мс
ЛОС Диаг.	357	$9.4 \cdot 10^{-21}$	17.81 мс

2.4 Большой тест 4545

Метод	Итераций	Относительная невязка	Время
МСГ	2006	$9.2 \cdot 10^{-21}$	417.82 мс
МСГ LU(sq)	11	$5.7 \cdot 10^{-21}$	11.67 мс
МСГ Диаг.	157	$9.8 \cdot 10^{-21}$	36.55 мс
ЛОС	2119	$9.7 \cdot 10^{-21}$	469.41 мс
ЛОС LU(sq)	9	$1.8 \cdot 10^{-21}$	6.79 мс
ЛОС Диаг.	1701	$9.7 \cdot 10^{-21}$	425.378 мс

2.5 Матрицы Гильберта

2.5.1 Размерность 5

Метод	Итераций	Относительная невязка	Время
МСГ	7	$4.6 \cdot 10^{-14}$	2.81 мкс
МСГ LU(sq)	1	$2.5 \cdot 10^{-14}$	2.3 мкс
МСГ Диаг.	7	$5.4 \cdot 10^{-15}$	3.32 мкс
ЛОС	7	$7.2 \cdot 10^{-14}$	10.72 мкс
ЛОС LU(sq)	1	$3.7 \cdot 10^{-14}$	4.24 мкс
ЛОС Диаг.	7	$1.2 \cdot 10^{-14}$	10.83 мкс

2.5.2 Размерность 10

Метод	Итераций	Относительная невязка	Время
МСГ	18	$1.6 \cdot 10^{-15}$	6.75 мкс
МСГ LU(sq)	2	$1.0 \cdot 10^{-15}$	4.25 мкс
МСГ Диаг.	17	$1.9 \cdot 10^{-16}$	7.49 мкс
ЛОС	17	$7.9 \cdot 10^{-15}$	6.61 мкс
ЛОС LU(sq)	2	$1.3 \cdot 10^{-15}$	3.25 мкс
ЛОС Диаг.	16	$8.8 \cdot 10^{-15}$	7.1 мкс

3 Выводы

- По таблицам видно, что использование предобусловливания, даже диагонального, положительно влияет на скорость сходимости.
- В среднем самый быстрый метод - ЛОС LU(sq).
- Диагональное предобуславливание увеличивает скорость сходимости для МСГ больше, чем для ЛОС.

4 Код программы

FILE **program.cpp**

```cpp
#include <vector>
#include <iostream>
#include <fstream>
#include <chrono>
#include <functional>
#include <sstream>
#include <iomanip>

#include ”../1/matrix.h”
#include ”../1/vector.h”

using namespace std;

//#define FULL_FACTORIZATION

//-----------------------------------------------------------------------------
ostream& operator<<(ostream& out, const vector<double>& x) {
    out.precision(16);
    out << ”(”;
    if (x.size() != 0) {
        for (int i = 0; i < x.size() - 1; ++i)
            out << x[i] << ”, ”;
        out << x.back() << ”)”;
    } else {
        out << ”)”;
    }
    return out;
}

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

//-----------------------------------------------------------------------------
double length(const vector<double>& mas) {
    return sqrt(mas*mas);
}

//-----------------------------------------------------------------------------
vector<double> to(const Vector& a) {
    vector<double> result(a.size());
    for (int i = 0; i < a.size(); ++i)
        result[i] = a(i);
    return result;
}

//-----------------------------------------------------------------------------
Vector to(const vector<double>& a) {
    Vector result(a.size());
    for (int i = 0; i < a.size(); ++i)
        result(i) = a[i];
    return result;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
struct matrix
{
    int n;
    vector<double> d, l, u;
    vector<int> i, j;

    void init(int n1) {
        n = n1;
        d.clear();
        l.clear();
        u.clear();
        i.clear();
        j.clear();
        d.resize(n);
        i.resize(n+1, 0);
    }

    void toDense(Matrix& m) const {
        m.resize(n, n, 0);
        for (int i = 0; i < n; ++i) {
            m(i, i) = d[i];
            for (int j = 0; j < lineElemCount(i); ++j) {
                m(i, lineElemRow(i, j)) = l[lineElemStart(i) + j];
                m(lineElemRow(i, j), i) = u[lineElemStart(i) + j];
            }
        }
    }

    int lineElemStart(int line) const {
        return i[line];
    }
    int lineStart(int line) const {
        return j[lineElemStart(line)];
    }
    int lineSize(int line) const {
        return line - lineStart(line);
    }
    int lineElemRow(int line, int elem) const {
        return j[lineElemStart(line) + elem];
    }
    int lineElemCount(int line) const {
        return i[line+1]-i[line];
    }
};

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

#ifdef FULL_FACTORIZATION

struct matrix_iterator
{
    matrix_iterator(const matrix& m, int line) : m(m), line(line) {
        if (m.lineElemCount(line) != 0) {
            pos = m.lineStart(line);
            j = m.lineElemStart(line);
            is_on_elem = true;
            is_empty_line = false;
            line_size = m.lineElemCount(line);
        } else {
            pos = 0;
            j = 0;
            is_on_elem = false;
            is_empty_line = true;
            line_size = 0;
        }
    }

    matrix_iterator& operator++() {
        if (!is_empty_line && j+1 < m.lineElemStart(line) + line_size && pos+1 ==
      ↪  m.lineElemRow(line, j+1-m.lineElemStart(line))) {
            is_on_elem = true;
            pos++;
            j++;
        } else {
            is_on_elem = false;
            pos++;
        }
        return *this;
    }

    int getpos(void) const { return pos; }
protected:
    bool is_on_elem, is_empty_line;
    const matrix& m;
    int line, line_size;
    int pos, j, end;
};

struct matrix_iterator_l : public matrix_iterator
{
    matrix_iterator_l(const matrix& m, int line) : matrix_iterator(m, line) {}
    double operator*() const {
        if (is_on_elem) return m.l[j];
        else return 0;
    }
};

struct matrix_iterator_u : public matrix_iterator
{
    matrix_iterator_u(const matrix& m, int row) : matrix_iterator(m, row) {}
    double operator*() const {
        if (is_on_elem) return m.u[j];
        else return 0;
    }
};

struct line_iterator
{
    line_iterator(const vector<pair<int, double>>& line) : line(line), i(0), pos(0),
  ↪  is_on_elem(line.size() != 0) {
        if (line.size() != 0) {
            pos = line[0].first;
            is_on_elem = true;
        }
    }

    line_iterator& operator++() {
        if (line.size() != 0 && pos+1 <= line.back().first && i + 1 < line.size() && pos+1 ==
      ↪  line[i+1].first) {
            is_on_elem = true;
            pos++;
            i++;
        } else {
            is_on_elem = false;
            pos++;
        }
        return *this;
    }

    double operator*() const {
        if (is_on_elem) return line[i].second;
        else return 0;
    }

    int getpos(void) const { return pos; }
protected:
    const vector<pair<int, double>>& line;
    int i, pos;
    bool is_on_elem;
};

template<class T1, class T2>
void synchronize_iterators(T1& i, T2& j) {
    while (i.getpos() != j.getpos()) {
        if (i.getpos() < j.getpos())
            ++i;
        else
            ++j;
    }
}

#endif

void lu_decompose(const matrix& a, matrix& lu) {
    #ifdef FULL_FACTORIZATION
    lu.init(a.n);
    for (int i = 0; i < a.n; ++i) {
        // Считаем элементы матрицы L
        vector<pair<int, double>> l_add, u_add;
        {
            matrix_iterator_l a_j(a, i);
            int iLineStart = a.lineStart(i);
            for (int j = iLineStart; j < i; ++j, ++a_j) {
                double sum = 0;
                if (j != iLineStart) {
                    line_iterator l_k(l_add);
                    matrix_iterator_u u_k(lu, j);
                    synchronize_iterators(l_k, u_k);
                    for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
                        sum += (*l_k) * (*u_k);
                }

                double res = ((*a_j) - sum) / lu.d[j];
                if (res != 0)
                    l_add.push_back({j, res});
            }
        }

        // Считаем элементы матрицы U
        {
            matrix_iterator_u a_j(a, i);
            int iRowStart = a.lineStart(i);
            for (int j = iRowStart; j < i; ++j, ++a_j) {
                double sum = 0;
                if (j != 0) {
                    matrix_iterator_l l_k(lu, j);
                    line_iterator u_k(u_add);
                    synchronize_iterators(l_k, u_k);
                    for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
                        sum += (*l_k) * (*u_k);
                }

                double res = ((*a_j) - sum) / lu.d[j];
                if (res != 0)
                    u_add.push_back({j, res});
            }
        }

        // Находим такие элементы, которые имеются в одном массиве, но нет в другом
        {
            int j = 0;
            while (j < l_add.size() || j < u_add.size()) {
                if (j >= u_add.size()) {
                    u_add.insert(u_add.begin() + j, {l_add[j].first, 0});
                } else
                if (j >= l_add.size()) {
                    l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
                } else
                if (l_add[j].first != u_add[j].first) {
                    if (u_add[j].first > l_add[j].first)
                        u_add.insert(u_add.begin() + j, {l_add[j].first, 0});
                    else
                        l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
                }
                j++;
            }
        }

        if (l_add.size() != u_add.size())
            throw std::exception();

        // Добавляем формат к обоим массивам
        lu.i[i+1] = lu.i[i] + l_add.size();
        for (int j = 0; j < l_add.size(); ++j) {
            lu.j.push_back(l_add[j].first);
            lu.l.push_back(l_add[j].second);
            lu.u.push_back(u_add[j].second);
        }

        // Считаем элементы диагонали
        {
            double sum = 0;
            if (i != 0) {
                for (int k = 0; k < l_add.size(); ++k)
                    sum += l_add[k].second * u_add[k].second;
            }

            double res = sqrt((a.d[i]) - sum);
            lu.d[i] = res;
        }
    }
    #endif

    #ifndef FULL_FACTORIZATION
    lu = a;
    for (int i = 0; i < lu.n; ++i) {
        // Заполняем нижний треугольник
        int line_start = lu.lineElemStart(i);
        int line_end = lu.lineElemStart(i+1);
        for (int j = line_start; j < line_end; ++j) {
            double sum = 0;

            int row = lu.j[j];
            int row_start = lu.lineElemStart(row);
            int row_end = lu.lineElemStart(row+1);

            int kl = line_start;
            int ku = row_start;

            while (kl < j && ku < row_end) {
                if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
                    sum += lu.l[kl] * lu.u[ku];
                    ku++;
                    kl++;
                } else if (lu.j[kl] < lu.j[ku]) {
                    kl++;
                } else {
                    ku++;
                }
            }

            lu.l[j] = (a.l[j] - sum) / lu.d[row];
        }

        // Заполняем верхний треугольник
        int row_start = lu.lineElemStart(i);
        int row_end = lu.lineElemStart(i+1);
        for (int j = line_start; j < line_end; ++j) {
            double sum = 0;

            int line = lu.j[j];
            int line_start = lu.lineElemStart(line);
            int line_end = lu.lineElemStart(line+1);

            int kl = line_start;
            int ku = row_start;

            while (kl < line_end && ku < j) {
                if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
                    sum += lu.l[kl] * lu.u[ku];
                    ku++;
                    kl++;
                } else if (lu.j[kl] < lu.j[ku]) {
                    kl++;
                } else {
                    ku++;
                }
            }

            lu.u[j] = (a.u[j] - sum) / lu.d[line];
        }

        // Расчитываем диагональный элемент
        double sum = 0;
        int line_row_start = lu.lineElemStart(i);
        int line_row_end = lu.lineElemStart(i+1);
        for (int j = line_row_start; j < line_row_end; ++j)
            sum += lu.l[j] * lu.u[j];

        lu.d[i] = sqrt(a.d[i] - sum);
    }
    #endif
}

//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
void mul(const matrix& a, vector<double>& x_y) {
    vector<double> result(a.n, 0);

    for (int i = 0; i < a.n; ++i) {
        int start = a.lineElemStart(i);
        int size = a.lineElemCount(i);
        for (int j = 0; j < size; j++) {
            result[i] += a.l[start + j] * x_y[a.lineElemRow(i, j)];
            result[a.lineElemRow(i, j)] += a.u[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < a.n; ++i)
        result[i] += a.d[i] * x_y[i];

    x_y = result;
}

//----------------------------------------------------------------------------
void mul_t(const matrix& a, vector<double>& x_y) {
    vector<double> result(a.n, 0);

    for (int i = 0; i < a.n; ++i) {
        int start = a.lineElemStart(i);
        int size = a.lineElemCount(i);
        for (int j = 0; j < size; j++) {
            result[i] += a.u[start + j] * x_y[a.lineElemRow(i, j)];
            result[a.lineElemRow(i, j)] += a.l[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < a.n; ++i)
        result[i] += a.d[i] * x_y[i];

    x_y = result;
}

//----------------------------------------------------------------------------
void mul_l_invert_t(const matrix& l, vector<double>& y_x) {
    for (int i = l.n - 1; i >= 0; i--) {
        int start = l.lineElemStart(i);
        int size = l.lineElemCount(i);

        y_x[i] /= l.d[i];
        for (int j = 0; j < size; ++j)
            y_x[l.lineElemRow(i, j)] -= y_x[i] * l.l[start + j];
    }
}

//----------------------------------------------------------------------------
void mul_u_invert_t(const matrix& u, vector<double>& y_x) {
    for (int i = 0; i < u.n; ++i) {
        int start = u.lineElemStart(i);
        int size = u.lineElemCount(i);

        sumreal sum = 0;
        for (int j = 0; j < size; ++j)
```

```cpp
            sum += u.u[start + j] * y_x[u.lineElemRow(i, j)];
        y_x[i] = (y_x[i] - sum) / u.d[i];
    }
}

//----------------------------------------------------------------------------
void mul_l_invert(const matrix& l, vector<double>& y_x) {
    for (int i = 0; i < l.n; ++i) {
        int start = l.lineElemStart(i);
        int size = l.lineElemCount(i);

        sumreal sum = 0;
        for (int j = 0; j < size; ++j)
            sum += l.l[start + j] * y_x[l.lineElemRow(i, j)];
        y_x[i] = (y_x[i] - sum) / l.d[i];
    }
}

//----------------------------------------------------------------------------
void mul_u_invert(const matrix& u, vector<double>& y_x) {
    for (int i = u.n-1; i >= 0; i--) {
        int start = u.lineElemStart(i);
        int size = u.lineElemCount(i);

        y_x[i] /= u.d[i];
        for (int j = 0; j < size; ++j)
            y_x[u.lineElemRow(i, j)] -= y_x[i] * u.u[start + j];
    }
}

//----------------------------------------------------------------------------
void mul_u(const matrix& u, vector<double>& x_y) {
    vector<double> result(u.n, 0);

    for (int i = 0; i < u.n; ++i) {
        int start = u.lineElemStart(i);
        int size = u.lineElemCount(i);
        for (int j = 0; j < size; j++) {
            result[u.lineElemRow(i, j)] += u.u[start + j] * x_y[i];
        }
    }

    // Умножение диагональных элементов на вектор
    for (int i = 0; i < u.n; ++i)
        result[i] += u.d[i] * x_y[i];

    x_y = result;
}

//----------------------------------------------------------------------------
void mul(const vector<double>& d, vector<double>& x_y) {
    for (int i = 0; i < d.size(); i++)
        x_y[i] *= d[i];
}

//----------------------------------------------------------------------------
void mul_invert(const vector<double>& d, vector<double>& x_y) {
    for (int i = 0; i < d.size(); i++)
        x_y[i] /= d[i];
}

//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
class SLAU
{
public:

//----------------------------------------------------------------------------
void read(string dir) {
    ifstream fin;

    fin.open(dir + ”/kuslau.txt”);
    fin >> n >> maxiter >> eps;
    fin.close();

    a.n = n;

    a.d.resize(n);
    fin.open(dir + ”/di.txt”);
    for (auto& i : a.d) fin >> i;
    fin.close();

    f.resize(n);
    fin.open(dir + ”/pr.txt”);
    for (auto& i : f) fin >> i;
    fin.close();

    a.i.resize(n+1);
    fin.open(dir + ”/ig.txt”);
    for (auto& i : a.i) { fin >> i; i--; }
    fin.close();

    a.j.resize(a.i.back());
    fin.open(dir + ”/jg.txt”);
    for (auto& i : a.j) { fin >> i; i--; }
    fin.close();

    a.l.resize(a.i.back());
    fin.open(dir + ”/ggl.txt”);
    for (auto& i : a.l) { fin >> i; }
    fin.close();

    a.u.resize(a.i.back());
    fin.open(dir + ”/ggu.txt”);
    for (auto& i : a.u) { fin >> i; }
    fin.close();

    x.resize(n);
    t1.resize(n);
    t2.resize(n);

    is_log = true;
}

//----------------------------------------------------------------------------
pair<int, double> msg1() {
    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; ++i)
        r[i] = f[i]-r[i];

    z = r;
    double rr = r*r;
    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        t1 = z;
        mul(a, t1);
        double alpha = (rr) / (t1*z);
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * t1[i];
        }
        double rr2 = r*r;
        double beta = rr2/rr;
        rr = rr2;
        for (int i = 0; i < n; ++i)
            z[i] = r[i] + beta * z[i];
        residual = sqrt(rr) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) << setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    return {i, residual};
}

//----------------------------------------------------------------------------
pair<int, double> msg2() {
    lu_decompose(a, lu);

    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; ++i)
        r[i] = f[i]-r[i];
    mul_l_invert(lu, r);
    mul_l_invert_t(lu, r);
    mul_t(a, r);
    mul_u_invert_t(lu, r);

    mul_u(lu, x);

    z = r;
    double rr = r*r;
    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        t1 = z;
        mul_u_invert(lu, t1);
        mul(a, t1);
        mul_l_invert(lu, t1);
        mul_l_invert_t(lu, t1);
        mul_t(a, t1);
        mul_u_invert_t(lu, t1);
        double alpha = (rr) / (t1*z);
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * t1[i];
        }
        double rr2 = r*r;
        double beta = rr2/rr;
        rr = rr2;
        for (int i = 0; i < n; ++i)
            z[i] = r[i] + beta * z[i];
        residual = sqrt(rr) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) <<
       ↪  setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    mul_u_invert(lu, x);

    return {i, residual};
}

//-----------------------------------------------------------------------------
pair<int, double> msg3() {
    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; ++i)
        r[i] = f[i]-r[i];

    z = r;
    mul_invert(a.d, z);

    double rr;
    t1 = r;
    mul_invert(a.d, t1);
    rr = t1*r;
    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        t1 = z;
        mul(a, t1);
        double alpha = (rr) / (t1*z);
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * t1[i];
        }
        t1 = r;
        mul_invert(a.d, t1);
        double rr2 = t1*r;
        double beta = rr2/rr;
        rr = rr2;
        for (int i = 0; i < n; ++i)
            z[i] = t1[i] + beta * z[i];
        residual = length(r) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) <<
       ↪  setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    return {i, residual};
}

//-----------------------------------------------------------------------------
pair<int, double> los1() {
    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; i++)
        r[i] = f[i] - r[i];

    z = r;

    p = z;
    mul(a, p);

    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        double pp = p*p;
        double alpha = (p*r) / pp;
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * p[i];
        }
        t1 = r;
        mul(a, t1);
        double beta = -(p*t1) / pp;
        for (int i = 0; i < n; ++i) {
            z[i] = r[i] + beta * z[i];
            p[i] = t1[i] + beta * p[i];
        }
        residual = length(r) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) <<
       ↪  setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    return {i, residual};
}

//-----------------------------------------------------------------------------
pair<int, double> los2() {
    lu_decompose(a, lu);
    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; i++)
        r[i] = f[i] - r[i];
    mul_l_invert(lu, r);

    z = r;
    mul_u_invert(lu, z);

    p = z;
    mul(a, p);
    mul_l_invert(lu, p);

    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        double pp = p*p;
        double alpha = (p*r) / pp;
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * p[i];
        }
        t1 = r;
        mul_u_invert(lu, t1);
        t2 = t1;
        mul(a, t2);
        mul_l_invert(lu, t2);
        double beta = -(p*t2) / pp;
        for (int i = 0; i < n; ++i) {
            z[i] = t1[i] + beta * z[i];
            p[i] = t2[i] + beta * p[i];
        }
        residual = length(r) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) <<
       ↪  setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    return {i, residual};
}

//-----------------------------------------------------------------------------
pair<int, double> los3() {
    x.clear();
    x.resize(n, 0);

    r = x;
    mul(a, r);
    for (int i = 0; i < n; i++)
        r[i] = f[i] - r[i];
    mul_invert(a.d, r);

    z = r;
    mul_invert(a.d, z);

    p = z;
    mul(a, p);
    mul_invert(a.d, p);

    double flen = sqrt(f*f);
    double residual;

    int i = 0;
    while (true) {
        double pp = p*p;
        double alpha = (p*r) / pp;
        for (int i = 0; i < n; ++i) {
            x[i] += alpha * z[i];
            r[i] -= alpha * p[i];
        }
        t1 = r;
        mul_invert(a.d, t1);
        t2 = t1;
        mul(a, t2);
        mul_invert(a.d, t2);
        double beta = -(p*t2) / pp;
        for (int i = 0; i < n; ++i) {
            z[i] = t1[i] + beta * z[i];
            p[i] = t2[i] + beta * p[i];
        }
        residual = length(r) / flen;
        i++;

        if (is_log) cout << ”Iteration: ” << setw(4) << i << ”, Residual: ” << setw(20) <<
       ↪  setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
    }

    return {i, residual};
}

int n, maxiter;
double eps;
matrix a, lu;
vector<double> f;
vector<double> r, z, p;
vector<double> x, t1, t2;
bool is_log;

};

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
void make_gilbert(int size) {
    Matrix g;
    generateGilbertMatrix(size, g);
    Vector x, y;
    x.generate(size);
    mul(g, x, y);

    string dir = ”gilbert” + to_string(size);
    system((”mkdir ” + dir).c_str());
    ofstream fout;
    fout.precision(16);

    fout.open(dir + ”/kuslau.txt”);
    fout << size << ” ” << 1000 << ” ” << 1e-13;
    fout.close();

    fout.open(dir + ”/di.txt”);
    for (int i = 0; i < size; ++i)
        fout << double(1.0)/double((i+1)+(i+1)-1) << ” ”;
    fout.close();

    fout.open(dir + ”/pr.txt”);
    for (int i = 0; i < size; ++i)
        fout << y(i) << ” ”;
    fout.close();

    fout.open(dir + ”/ig.txt”);
    int sum = 1;
    fout << sum << ” ”;
    for (int i = 0; i < size; ++i) {
        sum += i;
        fout << sum << ” ”;
    }
    fout.close();

    fout.open(dir + ”/jg.txt”);
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < i; ++j)
            fout << j+1 << ” ”;
    fout.close();

    fout.open(dir + ”/ggl.txt”);
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < i; ++j)
            fout << double(1.0)/double((i+1)+(j+1)-1) << ” ”;
    fout.close();

    fout.open(dir + ”/ggu.txt”);
    for (int i = 0; i < size; ++i)
        for (int j = 0; j < i; ++j)
            fout << double(1.0)/double((i+1)+(j+1)-1) << ” ”;
    fout.close();
}

//-----------------------------------------------------------------------------
void test(SLAU& s, string dir) {
    ofstream fout(dir + ”/test.txt”);
    vector<double> f1;
    Matrix m, l, u, a, sub;
    Vector pr(s.f.size()), pr1, f1_m(f1.size());
    pr = to(f1);
    s.a.toDense(m);

    lu_decompose(s.a, s.lu);
    matrix l_s = s.lu, u_s = s.lu;
    l_s.u.clear(); l_s.u.resize(l_s.l.size(), 0);
    u_s.l.clear(); u_s.l.resize(u_s.u.size(), 0);
    l_s.toDense(l);
    u_s.toDense(u);
    mul(l, u, a);
    a.negate();
    sum(m, a, sub);

    for (int i = 0; i < sub.height(); i++) {
        for (size_t j = 0; j < sub.width(); j++) {
            if (fabs(sub(i, j)) < 0.000001)
                sub(i, j) = 0;
        }
    }

    fout << ”a in dense:” << endl;
    m.save(fout);
    fout << endl;

    fout << ”l in dense:” << endl;
    l.save(fout);
    fout << endl;

    fout << ”u in dense:” << endl;
    u.save(fout);
    fout << endl;

    fout << ”a - l*u:” << endl;
    sub.save(fout);
    fout << endl;

    pr = to(s.f);
    f1 = s.f;
    mul(m, pr, pr1);
    mul(s.a, f1);
    fout << ”a * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    f1 = s.f;
    transpose(m);
    mul(m, pr, pr1);
    transpose(m);
    mul_t(s.a, f1);
    fout << ”a^t * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    f1 = s.f;
    mul(u, pr, pr1);
    mul_u(s.lu, f1);
    fout << ”u * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    mul(l, pr, pr1);
    f1 = to(pr1);
    pr1 = pr;
    mul_l_invert(s.lu, f1);
    fout << ”l^-1 * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    transpose(u);
    mul(u, pr, pr1);
    transpose(u);
    f1 = to(pr1);
    pr1 = pr;
    mul_u_invert_t(s.lu, f1);
    fout << ”u^-t * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    mul(u, pr, pr1);
    f1 = to(pr1);
    pr1 = pr;
    mul_u_invert(s.lu, f1);
    fout << ”u^-1 * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    pr = to(s.f);
    transpose(l);
    mul(l, pr, pr1);
    transpose(l);
    f1 = to(pr1);
    pr1 = pr;
    mul_l_invert_t(s.lu, f1);
    fout << ”l^-t * vec:” << endl;
    fout << to(pr1) << endl;
    fout << f1 << endl;
    fout << endl;

    fout.close();
}

//------------------------------------------------------------------------------
string print_time(double time) {
    stringstream sout;
    if (time > 1000 * 1000) {
        sout << time / (1000 * 1000) << ” s”;
    } else if (time > 1000) {
        sout << time / (1000) << ” ms”;
    } else {
        sout << time << ” us”;
    }
    return sout.str();
}

//------------------------------------------------------------------------------
void test_method(string name, function<pair<int, double>(SLAU*)> f, SLAU& s, bool is_write_each_iteration) {
    cout << name << ”:” << endl;

    s.is_log = false;
    int count = 100;
    double time = 0;
    pair<int, double> temp_res;
    chrono::high_resolution_clock::time_point t1, t2;
    for (int i = 0; i < count; i++) {
        t1 = chrono::high_resolution_clock::now();
        temp_res = f(&s);
        t2 = chrono::high_resolution_clock::now();
        time += chrono::duration_cast<chrono::microseconds>(t2 - t1).count();
        if (time / 1000.0 > 1000.0) count = i+1;
    }
    time /= count;

    if (is_write_each_iteration) {
        s.is_log = true;
        f(&s);
    } else {
        cout << ”Iterations: ” << temp_res.first << ”, Residual: ” << temp_res.second << endl;
    }

    cout << ”Time: ” << print_time(time) << endl;
    cout << ”X: ” << s.x << endl << endl;
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

int main() {
    //for (int i = 0; i < 16; ++i) make_gilbert(i);

    string dir = ”test1”;
    bool is_write_to_file = true;
    bool is_write_each_iteration = false;
    bool is_write_tests = false;

    cout << ”Enter dir: ”;
    cin >> dir;
    cout << ”Is write to file? (0 or 1): ”;
    cin >> is_write_to_file;
    cout << ”Is write each iteration? (0 or 1): ”;
    cin >> is_write_each_iteration;
    cout << ”Is write tests? (0 or 1): ”;
    cin >> is_write_tests;

    if (is_write_to_file) freopen((dir + ”/res.txt”).c_str(), ”w”, stdout);

    SLAU s;
    s.read(dir);

    if (is_write_tests) test(s, dir);

    test_method(”MSG”, &SLAU::msg1, s, is_write_each_iteration);
    test_method(”MSG LUsq”, &SLAU::msg2, s, is_write_each_iteration);
    test_method(”MSG D”, &SLAU::msg3, s, is_write_each_iteration);
    test_method(”LOS”, &SLAU::los1, s, is_write_each_iteration);
    test_method(”LOS LUsq”, &SLAU::los2, s, is_write_each_iteration);
    test_method(”LOS D”, &SLAU::los3, s, is_write_each_iteration);

    if (!is_write_to_file) system(”pause”);
}
```