

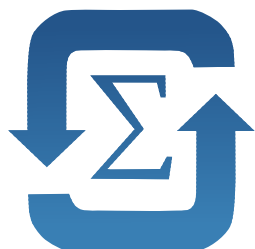
Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Кафедра прикладной математики

Лабораторная работа №3
по дисциплине «Численные методы»

Решение разреженных СЛАУ трехшаговыми итерационными
методами с предобуславливанием



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 предобуславливания 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

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

$$\varepsilon = 10^{-14}, \quad iterations_{max} = 10000, \quad start = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)^T$$

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

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

$$\varepsilon = 10^{-14}, \quad iterations_{max} = 10000, \quad start = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)^T$$

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

```
1 #include <vector>
2 #include <iostream>
3 #include <fstream>
4 #include <chrono>
5 #include <functional>
6 #include <sstream>
7 #include <iomanip>
8
9 #include "../1/matrix.h"
10 #include "../1/vector.h"
11
12 using namespace std;
13
14 #define FULL_FACTORIZATION
15
16 //-----
17 ostream& operator<<(ostream& out, const vector<double>& x) {
18     out.precision(16);
19     out << "x: ";
20     if (x.size() != 0) {
21         for (int i = 0; i < x.size() - 1; ++i)
22             out << x[i] << " ";
23         out << x.back() << " ";
24     } else {
25         out << " ";
26     }
27     return out;
28 }
29
30 //-----
31 double operator*(const vector<double>& a, const vector<double>& b) {
32     double sum = 0;
33     for (int i = 0; i < a.size(); ++i)
34         sum += a[i] * b[i];
35     return sum;
36 }
37
38 //-----
39 double length(const vector<double>& mas) {
40     return sqrt(mas*mas);
41 }
42
43 //-----
44 vector<double> to(const Vector& a) {
45     vector<double> result(a.size());
46     for (int i = 0; i < a.size(); ++i)
47         result[i] = a[i];
48     return result;
49 }
50
51 //-----
52 Vector to(const vector<double>& a) {
53     Vector result(a.size());
54     for (int i = 0; i < a.size(); ++i)
55         result[i] = a[i];
56     return result;
57 }
58
59 //-----
60 //-----
61 //-----
62
63 struct matrix
64 {
65     int n;
66     vector<double> d, l, u;
67     vector<int> i, j;
68
69     void init(int n1) {
70         n = n1;
71         d.clear();
72         l.clear();
73         u.clear();
74         i.clear();
75         j.clear();
76         d.resize(n);
77         i.resize(n+1, 0);
78     }
79
80     void toDense(Matrix& m) const {
81         m.resize(n, n, 0);
82         for (int i = 0; i < n; ++i) {
83             m[i, i] = d[i];
84             for (int j = 0; j < lineElemCount(i); ++j) {
85                 m[i, lineElemRow(i, j)] = l[lineElemStart(i) + j];
86                 m[lineElemRow(i, j), i] = u[lineElemStart(i) + j];
87             }
88         }
89     }
90
91     int lineElemStart(int line) const {
92         return i[line];
93     }
94
95     int lineStart(int line) const {
96         return j[lineElemStart(line)];
97     }
98
99     int lineSize(int line) const {
100         return line - lineStart(line);
101     }
102
103     int lineElemRow(int line, int elem) const {
104         return j[lineElemStart(line) + elem];
105     }
106
107     int lineElemCount(int line) const {
108         return i[line+1] - i[line];
109     }
110
111     //-----
112     //-----
113     //-----
114
115 #ifndef FULL_FACTORIZATION
116 struct matrix_iterator
117 {
118     matrix_iterator(const matrix& m, int line) : m(m), line(line) {
119         if (m.lineElemCount(line) != 0) {
120             pos = m.lineStart(line);
121             is_on_elem = true;
122             is_empty_line = false;
123             line_size = m.lineElemCount(line);
124         } else {
125             pos = 0;
126             j = 0;
127             is_on_elem = false;
128             is_empty_line = true;
129             line_size = 0;
130         }
131     }
132
133     matrix_iterator& operator++() {
134         if (!is_empty_line && j+1 < m.lineElemStart(line) + line_size && pos+1 ==
135             m.lineElemRow(line, j+1-m.lineElemStart(line))) {
136             is_on_elem = true;
137             pos++;
138             j++;
139         } else {
140             is_on_elem = false;
141             pos++;
142         }
143         return *this;
144     }
145
146     int getpos(void) const { return pos; }
147
148 protected:
149     bool is_on_elem, is_empty_line;
150     const matrix& m;
151     int line, line_size;
152     int pos, j, end;
153 };
154
155 struct matrix_iterator_l : public matrix_iterator
156 {
157     matrix_iterator_l(const matrix& m, int line) : matrix_iterator(m, line) {}
158     double operator*() const {
159         if (is_on_elem) return m.l[j];
160         else return 0;
161     }
162 };
163
164 struct matrix_iterator_u : public matrix_iterator
165 {
166     matrix_iterator_u(const matrix& m, int row) : matrix_iterator(m, row) {}
167     double operator*() const {
168         if (is_on_elem) return m.u[j];
169         else return 0;
170     }
171 };
172
173 struct line_iterator
174 {
175     line_iterator(const vector<pair<int, double>>& line) : line(line), i(0), pos(0),
176         is_on_elem(line.size() != 0) {
177         if (line.size() != 0) {
178             pos = line[0].first;
179             is_on_elem = true;
180         }
181     }
182
183     line_iterator& operator++() {
184         if (line.size() != 0 && pos+1 <= line.back().first && i+1 < line.size() && pos+1 ==
185             line[i+1].first) {
186             is_on_elem = true;
187             pos++;
188             i++;
189         } else {
190             is_on_elem = false;
191             pos++;
192         }
193         return *this;
194     }
195
196     double operator*() const {
197         if (is_on_elem) return line[i].second;
198         else return 0;
199     }
200
201     int getpos(void) const { return pos; }
202
203 protected:
204     const vector<pair<int, double>>& line;
205     int i, pos;
206     bool is_on_elem;
207 };
208
209 template<class T1, class T2>
210 void synchronize_iterators(T1& i, T2& j) {
211     while (i.getpos() != j.getpos()) {
212         if (i.getpos() < j.getpos())
213             ++i;
214         else
215             ++j;
216     }
217 }
218
219 #endif
220
221 void lu_decompose(const matrix& a, matrix& lu) {
222     #ifndef FULL_FACTORIZATION
223     lu.init(a.n);
224     for (int i = 0; i < a.n; ++i) {
225         // Считаем элементы матрицы L
226         vector<pair<int, double>> l_add, u_add;
227         {
228             matrix_iterator_l a_j(a, i);
229             int iLineStart = a.lineStart(i);
230             for (int j = iLineStart; j < i; ++j, ++a_j) {
231                 double sum = 0;
232                 if (j != iLineStart) {
233                     line_iterator_l k(l_add);
234                     matrix_iterator_u u_k(lu, j);
235                     synchronize_iterators(k, u_k);
236                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
237                         sum += (*l_k) * (*u_k);
238                 }
239                 double res = ((*a_j) - sum) / lu.d[j];
240                 if (res != 0)
241                     l_add.push_back({j, res});
242             }
243         }
244         // Считаем элементы матрицы U
245         {
246             matrix_iterator_u a_j(a, i);
247             int iRowStart = a.lineStart(i);
248             for (int j = iRowStart; j < i; ++j, ++a_j) {
249                 double sum = 0;
250                 if (j != 0) {
251                     matrix_iterator_l k(lu, j);
252                     line_iterator u_k(u_add);
253                     synchronize_iterators(k, u_k);
254                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
255                         sum += (*l_k) * (*u_k);
256                 }
257                 double res = ((*a_j) - sum) / lu.d[j];
258                 if (res != 0)
259                     u_add.push_back({j, res});
260             }
261         }
262         // Находим такие элементы, которые имеются в одном массиве, но нет в другом
263     }
264 }
```

```
130 }
131
132 matrix_iterator& operator++() {
133     if (!is_empty_line && j+1 < m.lineElemStart(line) + line_size && pos+1 ==
134         m.lineElemRow(line, j+1-m.lineElemStart(line))) {
135         is_on_elem = true;
136         pos++;
137         j++;
138     } else {
139         is_on_elem = false;
140         pos++;
141     }
142     return *this;
143 }
144
145 int getpos(void) const { return pos; }
146
147 protected:
148     bool is_on_elem, is_empty_line;
149     const matrix& m;
150     int line, line_size;
151     int pos, j, end;
152 };
153
154 struct matrix_iterator_l : public matrix_iterator
155 {
156     matrix_iterator_l(const matrix& m, int line) : matrix_iterator(m, line) {}
157     double operator*() const {
158         if (is_on_elem) return m.l[j];
159         else return 0;
160     }
161 };
162
163 struct matrix_iterator_u : public matrix_iterator
164 {
165     matrix_iterator_u(const matrix& m, int row) : matrix_iterator(m, row) {}
166     double operator*() const {
167         if (is_on_elem) return m.u[j];
168         else return 0;
169     }
170 };
171
172 struct line_iterator
173 {
174     line_iterator(const vector<pair<int, double>>& line) : line(line), i(0), pos(0),
175         is_on_elem(line.size() != 0) {
176         if (line.size() != 0) {
177             pos = line[0].first;
178             is_on_elem = true;
179         }
180     }
181
182     line_iterator& operator++() {
183         if (line.size() != 0 && pos+1 <= line.back().first && i+1 < line.size() && pos+1 ==
184             line[i+1].first) {
185             is_on_elem = true;
186             pos++;
187             i++;
188         } else {
189             is_on_elem = false;
190             pos++;
191         }
192         return *this;
193     }
194
195     double operator*() const {
196         if (is_on_elem) return line[i].second;
197         else return 0;
198     }
199
200     int getpos(void) const { return pos; }
201
202 protected:
203     const vector<pair<int, double>>& line;
204     int i, pos;
205     bool is_on_elem;
206 };
207
208 template<class T1, class T2>
209 void synchronize_iterators(T1& i, T2& j) {
210     while (i.getpos() != j.getpos()) {
211         if (i.getpos() < j.getpos())
212             ++i;
213         else
214             ++j;
215     }
216 }
217
218 #endif
219
220 void lu_decompose(const matrix& a, matrix& lu) {
221     #ifndef FULL_FACTORIZATION
222     lu.init(a.n);
223     for (int i = 0; i < a.n; ++i) {
224         // Считаем элементы матрицы L
225         vector<pair<int, double>> l_add, u_add;
226         {
227             matrix_iterator_l a_j(a, i);
228             int iLineStart = a.lineStart(i);
229             for (int j = iLineStart; j < i; ++j, ++a_j) {
230                 double sum = 0;
231                 if (j != iLineStart) {
232                     line_iterator_l k(l_add);
233                     matrix_iterator_u u_k(lu, j);
234                     synchronize_iterators(k, u_k);
235                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
236                         sum += (*l_k) * (*u_k);
237                 }
238                 double res = ((*a_j) - sum) / lu.d[j];
239                 if (res != 0)
240                     l_add.push_back({j, res});
241             }
242         }
243         // Считаем элементы матрицы U
244         {
245             matrix_iterator_u a_j(a, i);
246             int iRowStart = a.lineStart(i);
247             for (int j = iRowStart; j < i; ++j, ++a_j) {
248                 double sum = 0;
249                 if (j != 0) {
250                     matrix_iterator_l k(lu, j);
251                     line_iterator u_k(u_add);
252                     synchronize_iterators(k, u_k);
253                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
254                         sum += (*l_k) * (*u_k);
255                 }
256                 double res = ((*a_j) - sum) / lu.d[j];
257                 if (res != 0)
258                     u_add.push_back({j, res});
259             }
260         }
261         // Находим такие элементы, которые имеются в одном массиве, но нет в другом
262     }
263 }
```

```

263     int j = 0;
264     while (j < l_add.size() || j < u_add.size()) {
265         if (j >= u_add.size()) {
266             u_add.insert(u_add.begin() + j, {l_add[j].first, 0});
267         } else
268             if (j >= l_add.size()) {
269                 l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
270             } else
271                 if (l_add[j].first != u_add[j].first) {
272                     if (u_add[j].first > l_add[j].first)
273                         u_add.insert(u_add.begin() + j, {l_add[j].first, 0});
274                     else
275                         l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
276                 }
277             j++;
278     }
279
280     if (l_add.size() != u_add.size())
281         throw std::exception();
282
283     // Добавляем формат к обоим массивам
284     lu.l[i+1] = lu.l[i] + l_add.size();
285     for (int j = 0; j < l_add.size(); ++j) {
286         lu.j.push_back(l_add[j].first);
287         lu.l.push_back(l_add[j].second);
288     }
289 }
290
291 // Считаем элементы диагонали
292 double sum = 0;
293 if (i != 0) {
294     for (int k = 0; k < l_add.size(); ++k)
295         sum += l_add[k].second * u_add[k].second;
296 }
297 double res = sqrt((a.d[i] - sum));
298 lu.d[i] = res;
299 }
300 }
301
302 #endif
303
304 #ifndef FULL_FACTORIZATION
305 lu = a;
306 for (int i = 0; i < lu.n; ++i) {
307     // Заполняем нижний треугольник
308     int line_start = lu.lineElemStart(i);
309     int line_end = lu.lineElemStart(i+1);
310     for (int j = line_start; j < line_end; ++j) {
311         double sum = 0;
312
313         int row = lu.j[j];
314         int row_start = lu.lineElemStart(row);
315         int row_end = lu.lineElemStart(row+1);
316
317         int kl = line_start;
318         int ku = row_start;
319
320         while (kl < j && ku < row_end) {
321             if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
322                 sum += lu.l[kl] * lu.u[ku];
323                 ku++;
324             } else if (lu.j[kl] < lu.j[ku]) {
325                 kl++;
326             } else {
327                 ku++;
328             }
329         }
330         lu.l[j] = (a.l[j] - sum) / lu.d[row];
331     }
332 }
333
334 // Заполняем верхний треугольник
335 int row_start = lu.lineElemStart(i);
336 int row_end = lu.lineElemStart(i+1);
337 for (int j = line_start; j < line_end; ++j) {
338     double sum = 0;
339
340     int line = lu.j[j];
341     int line_start = lu.lineElemStart(line);
342     int line_end = lu.lineElemStart(line+1);
343
344     int kl = line_start;
345     int ku = row_start;
346
347     while (kl < line_end && ku < j) {
348         if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
349             sum += lu.l[kl] * lu.u[ku];
350             ku++;
351         } else if (lu.j[kl] < lu.j[ku]) {
352             kl++;
353         } else {
354             ku++;
355         }
356     }
357     lu.u[j] = (a.u[j] - sum) / lu.d[line];
358 }
359
360 // Расчитываем диагональный элемент
361 double sum = 0;
362 int line_row_start = lu.lineElemStart(i);
363 int line_row_end = lu.lineElemStart(i+1);
364 for (int j = line_row_start; j < line_row_end; ++j)
365     sum += lu.l[j] * lu.u[j];
366 lu.d[i] = sqrt(a.d[i] - sum);
367 }
368 #endif
369
370 //-----
371 //-----
372 //-----
373
374 void mul(const matrix& a, vector<double>& x_y) {
375     vector<double> result(a.n, 0);
376
377     for (int i = 0; i < a.n; ++i) {
378         int start = a.lineElemStart(i);
379         int size = a.lineElemCount(i);
380         for (int j = 0; j < size; ++j) {
381             result[i] += a.l[start + j] * x_y[a.lineElemRow(i, j)];
382             result[a.lineElemRow(i, j)] += a.u[start + j] * x_y[i];
383         }
384     }
385
386     // Умножение диагональных элементов на вектор
387     for (int i = 0; i < a.n; ++i)
388         result[i] += a.d[i] * x_y[i];
389
390     x_y = result;
391 }
392
393 void mul_t(const matrix& a, vector<double>& x_y) {
394     vector<double> result(a.n, 0);
395
396     for (int i = 0; i < a.n; ++i) {
397         int start = a.lineElemStart(i);
398         int size = a.lineElemCount(i);
399         for (int j = 0; j < size; ++j) {
400             result[i] += a.u[start + j] * x_y[a.lineElemRow(i, j)];
401             result[a.lineElemRow(i, j)] += a.l[start + j] * x_y[i];
402         }
403     }
404
405     // Умножение диагональных элементов на вектор
406     for (int i = 0; i < a.n; ++i)
407         result[i] += a.d[i] * x_y[i];
408
409     x_y = result;
410 }
411
412 void mul_l_invert_t(const matrix& l, vector<double>& y_x) {
413     for (int i = l.n - 1; i >= 0; i--) {
414         int start = l.lineElemStart(i);
415         int size = l.lineElemCount(i);
416
417         y_x[i] /= l.d[i];
418         for (int j = 0; j < size; ++j)
419             y_x[l.lineElemRow(i, j)] -= y_x[i] * l.l[start + j];
420     }
421 }
422
423 void mul_u_invert_t(const matrix& u, vector<double>& y_x) {
424     for (int i = 0; i < u.n; ++i) {
425         int start = u.lineElemStart(i);
426         int size = u.lineElemCount(i);
427
428         sumreal sum = 0;
429         for (int j = 0; j < size; ++j)
430
441         sum += u.u[start + j] * y_x[u.lineElemRow(i, j)];
442         y_x[i] = (y_x[i] - sum) / u.d[i];
443     }
444 }
445
446 //-----
447 void mul_l_invert(const matrix& l, vector<double>& y_x) {
448     for (int i = 0; i < l.n; ++i) {
449         int start = l.lineElemStart(i);
450         int size = l.lineElemCount(i);
451
452         sumreal sum = 0;
453         for (int j = 0; j < size; ++j)
454             sum += l.l[start + j] * y_x[l.lineElemRow(i, j)];
455         y_x[i] = (y_x[i] - sum) / l.d[i];
456     }
457 }
458
459 //-----
460 void mul_u_invert(const matrix& u, vector<double>& y_x) {
461     for (int i = u.n - 1; i >= 0; i--) {
462         int start = u.lineElemStart(i);
463         int size = u.lineElemCount(i);
464
465         y_x[i] /= u.d[i];
466         for (int j = 0; j < size; ++j)
467             y_x[u.lineElemRow(i, j)] -= y_x[i] * u.u[start + j];
468     }
469 }
470
471 //-----
472 void mul_u(const matrix& u, vector<double>& x_y) {
473     vector<double> result(u.n, 0);
474
475     for (int i = 0; i < u.n; ++i) {
476         int start = u.lineElemStart(i);
477         int size = u.lineElemCount(i);
478         for (int j = 0; j < size; ++j) {
479             result[u.lineElemRow(i, j)] += u.u[start + j] * x_y[i];
480         }
481     }
482
483     // Умножение диагональных элементов на вектор
484     for (int i = 0; i < u.n; ++i)
485         result[i] += u.d[i] * x_y[i];
486
487     x_y = result;
488 }
489
490 //-----
491 void mul(const vector<double>& d, vector<double>& x_y) {
492     for (int i = 0; i < d.size(); i++)
493         x_y[i] *= d[i];
494 }
495
496 //-----
497 void mul_invert(const vector<double>& d, vector<double>& x_y) {
498     for (int i = 0; i < d.size(); i++)
499         x_y[i] /= d[i];
500 }
501
502 //-----
503 //-----
504 //-----
505
506 //----- SLAU
507 class SLAU {
508 public:
509
510     //-----
511     void read(string dir) {
512         ifstream fin;
513         fin.open(dir + "/kuslau.txt");
514         fin >> n >> maxiter >> eps;
515         fin.close();
516
517         a.n = n;
518
519         a.d.resize(n);
520         fin.open(dir + "/di.txt");
521         for (auto& i : a.d) fin >> i;
522         fin.close();
523
524         f.resize(n);
525         fin.open(dir + "/pr.txt");
526         for (auto& i : f) fin >> i;
527         fin.close();
528
529         a.i.resize(n+1);
530         fin.open(dir + "/ig.txt");
531         for (auto& i : a.i) { fin >> i; i--; }
532         fin.close();
533
534         a.j.resize(a.i.back());
535         fin.open(dir + "/jg.txt");
536         for (auto& i : a.j) { fin >> i; i--; }
537         fin.close();
538
539         a.l.resize(a.i.back());
540         fin.open(dir + "/lg.txt");
541         for (auto& i : a.l) { fin >> i; }
542         fin.close();
543
544         a.u.resize(a.i.back());
545         fin.open(dir + "/gu.txt");
546         for (auto& i : a.u) { fin >> i; }
547         fin.close();
548
549         x.resize(n);
550         t1.resize(n);
551         t2.resize(n);
552
553         is_log = true;
554     }
555
556     //-----
557     pair<int, double> msg1() {
558         x.clear();
559         x.resize(n, 0);
560
561         r = x;
562         mul(a, r);
563         for (int i = 0; i < n; ++i)
564             r[i] = f[i] - r[i];
565
566         z = r;
567         double rr = r*r;
568         double flen = sqrt(f*f);
569         double residual;
570
571         int i = 0;
572         while (true) {
573             t1 = z;
574             mul(a, t1);
575             double alpha = (rr) / (t1*t2);
576             for (int i = 0; i < n; ++i) {
577                 x[i] += alpha * z[i];
578                 r[i] -= alpha * t1[i];
579             }
580             double rr2 = r*r;
581             double beta = rr2/rr;
582             for (int i = 0; i < n; ++i)
583                 z[i] = r[i] + beta * z[i];
584             residual = sqrt(rr) / flen;
585             i++;
586
587             if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
588                 << setprecision(16) << residual << endl;
589             if (fabs(residual) < eps || i > maxiter)
590                 break;
591         }
592
593         return {i, residual};
594     }
595
596     //-----
597     pair<int, double> msg2() {
598         lu_decompose(a, lu);
599
600         x.clear();
601         x.resize(n, 0);
602
603         r = x;
604         mul(a, r);
605         for (int i = 0; i < n; ++i)
606             r[i] = f[i] - r[i];
607
608         mul_l_invert(lu, r);
609         mul_l_invert_t(lu, r);
610         mul_t(a, r);
611         mul_u_invert_t(lu, r);
612         mul_u(lu, x);
613
614         z = r;
615     }
616 }

```

```

617 double rr = r*r;
618 double flen = sqrt(f*f);
619 double residual;
620
621 int i = 0;
622 while (true) {
623     t1 = z;
624     mul_u_invert(lu, t1);
625     mul(a, t1);
626     mul_l_invert(lu, t1);
627     mul_l_invert_t(lu, t1);
628     mul_t(a, t1);
629     mul_u_invert_t(lu, t1);
630     double alpha = (rr) / (t1*z);
631     for (int i = 0; i < n; ++i) {
632         x[i] += alpha * z[i];
633         r[i] -= alpha * t1[i];
634     }
635     double rr2 = r*r;
636     double beta = rr2/rr;
637     rr = rr2;
638     for (int i = 0; i < n; ++i)
639         z[i] = r[i] + beta * z[i];
640     residual = sqrt(rr) / flen;
641     i++;
642
643     if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
        << setprecision(16) << residual << endl;
        if (fabs(residual) < eps || i > maxiter)
            break;
644 }
645 mul_u_invert(lu, x);
646 return {i, residual};
647 }
648
649 //-----
650 pair<int, double> msg3() {
651     x.clear();
652     x.resize(n, 0);
653
654     r = x;
655     mul(a, r);
656     for (int i = 0; i < n; ++i)
657         r[i] = f[i] - r[i];
658
659     z = r;
660     mul_invert(a.d, z);
661
662     double rr;
663     t1 = r;
664     mul_invert(a.d, t1);
665     rr = t1*r;
666     double flen = sqrt(f*f);
667     double residual;
668
669     int i = 0;
670     while (true) {
671         t1 = z;
672         mul(a, t1);
673         double alpha = (rr) / (t1*z);
674         for (int i = 0; i < n; ++i) {
675             x[i] += alpha * z[i];
676             r[i] -= alpha * t1[i];
677         }
678         t1 = r;
679         mul_invert(a.d, t1);
680         double rr2 = t1*r;
681         double beta = rr2/rr;
682         rr = rr2;
683         for (int i = 0; i < n; ++i)
684             z[i] = t1[i] + beta * z[i];
685         residual = length(r) / flen;
686         i++;
687
688         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
            << setprecision(16) << residual << endl;
            if (fabs(residual) < eps || i > maxiter)
                break;
689     }
690     return {i, residual};
691 }
692
693 //-----
694 pair<int, double> los1() {
695     x.clear();
696     x.resize(n, 0);
697
698     r = x;
699     mul(a, r);
700     for (int i = 0; i < n; ++i)
701         r[i] = f[i] - r[i];
702
703     z = r;
704
705     p = z;
706     mul(a, p);
707
708     double flen = sqrt(f*f);
709     double residual;
710
711     int i = 0;
712     while (true) {
713         double pp = p*p;
714         double alpha = (p*r) / pp;
715         for (int i = 0; i < n; ++i) {
716             x[i] += alpha * z[i];
717             r[i] -= alpha * p[i];
718         }
719         t1 = r;
720         mul(a, t1);
721         double beta = -(p*t1) / pp;
722         for (int i = 0; i < n; ++i) {
723             z[i] = r[i] + beta * z[i];
724             p[i] = t1[i] + beta * p[i];
725         }
726         residual = length(r) / flen;
727         i++;
728
729         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
            << setprecision(16) << residual << endl;
            if (fabs(residual) < eps || i > maxiter)
                break;
730     }
731     return {i, residual};
732 }
733
734 //-----
735 pair<int, double> los2() {
736     lu_decompose(a, lu);
737     x.clear();
738     x.resize(n, 0);
739
740     r = x;
741     mul(a, r);
742     for (int i = 0; i < n; ++i)
743         r[i] = f[i] - r[i];
744     mul_l_invert(lu, r);
745
746     z = r;
747     mul_u_invert(lu, z);
748
749     p = z;
750     mul(a, p);
751     mul_l_invert(lu, p);
752
753     double flen = sqrt(f*f);
754     double residual;
755
756     int i = 0;
757     while (true) {
758         double pp = p*p;
759         double alpha = (p*r) / pp;
760         for (int i = 0; i < n; ++i) {
761             x[i] += alpha * z[i];
762             r[i] -= alpha * p[i];
763         }
764         t1 = r;
765         mul_u_invert(lu, t1);
766         t2 = t1;
767         mul(a, t2);
768         mul_l_invert(lu, t2);
769         double beta = -(p*t2) / pp;
770         for (int i = 0; i < n; ++i) {
771             z[i] = t1[i] + beta * z[i];
772             p[i] = t2[i] + beta * p[i];
773         }
774         residual = length(r) / flen;
775         i++;
776
777         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
            << setprecision(16) << residual << endl;
            if (fabs(residual) < eps || i > maxiter)
                break;
778     }
779 }
780
781 break;
782 }
783
784 return {i, residual};
785 }
786
787 pair<int, double> los3() {
788     x.clear();
789     x.resize(n, 0);
790
791     r = x;
792     mul(a, r);
793     for (int i = 0; i < n; ++i)
794         r[i] = f[i] - r[i];
795     mul_invert(a.d, r);
796
797     z = r;
798     mul_invert(a.d, z);
799
800     p = z;
801     mul(a, p);
802     mul_invert(a.d, p);
803
804     double flen = sqrt(f*f);
805     double residual;
806
807     int i = 0;
808     while (true) {
809         double pp = p*p;
810         double alpha = (p*r) / pp;
811         for (int i = 0; i < n; ++i) {
812             x[i] += alpha * z[i];
813             r[i] -= alpha * p[i];
814         }
815         t1 = r;
816         mul_invert(a.d, t1);
817         t2 = t1;
818         mul(a, t2);
819         mul_invert(a.d, t2);
820         double beta = -(p*t2) / pp;
821         for (int i = 0; i < n; ++i) {
822             z[i] = t1[i] + beta * z[i];
823             p[i] = t2[i] + beta * p[i];
824         }
825         residual = length(r) / flen;
826         i++;
827
828         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
            << setprecision(16) << residual << endl;
            if (fabs(residual) < eps || i > maxiter)
                break;
829     }
830     return {i, residual};
831 }
832
833 int n, maxiter;
834 double eps;
835 matrix a, lu;
836 vector<double> f;
837 vector<double> r, z, p;
838 vector<double> x, t1, t2;
839 bool is_log;
840
841 //-----
842 //-----
843 //-----
844
845 void make_gilbert(int size) {
846     Matrix g;
847     generateGilbertMatrix(size, g);
848     Vector x, y;
849     x.generate(size);
850     mul(g, x, y);
851
852     string dir = "gilbert" + to_string(size);
853     system(("mkdir " + dir).c_str());
854     ofstream fout;
855     fout.precision(16);
856
857     fout.open(dir + "/kuslau.txt");
858     fout << size << " " << 1000 << " " << 1e-13;
859     fout.close();
860
861     fout.open(dir + "/di.txt");
862     for (int i = 0; i < size; ++i)
863         fout << double(1.0)/double((i+1)*(i+1)-1) << " ";
864     fout.close();
865
866     fout.open(dir + "/pr.txt");
867     for (int i = 0; i < size; ++i)
868         fout << y[i] << " ";
869     fout.close();
870
871     fout.open(dir + "/lg.txt");
872     int sum = 1;
873     fout << sum << " ";
874     for (int i = 0; i < size; ++i) {
875         sum += i;
876         fout << sum << " ";
877     }
878     fout.close();
879
880     fout.open(dir + "/jg.txt");
881     for (int i = 0; i < size; ++i)
882         for (int j = 0; j < i; ++j)
883             fout << j+1 << " ";
884     fout.close();
885
886     fout.open(dir + "/agl.txt");
887     for (int i = 0; i < size; ++i)
888         for (int j = 0; j < i; ++j)
889             fout << double(1.0)/double((i+1)*(j+1)-1) << " ";
890     fout.close();
891
892     fout.open(dir + "/ggg.txt");
893     for (int i = 0; i < size; ++i)
894         for (int j = 0; j < i; ++j)
895             fout << double(1.0)/double((i+1)*(j+1)-1) << " ";
896     fout.close();
897
898 //-----
899 void test(SLAW s, string dir) {
900     ofstream fout(dir + "/test.txt");
901     vector<double> f1;
902     Matrix m, l, u, a, sub;
903     Vector pr(s.f.size()), pr1, fl_m(f1.size());
904     pr = to(f1);
905     s.a.toDense(m);
906
907     lu_decompose(s.a, s.lu);
908     matrix l_s = s.lu, u_s = s.lu;
909     l_s.u.clear(); l_s.u.resize(l_s.l.size(), 0);
910     u_s.l.clear(); u_s.l.resize(u_s.u.size(), 0);
911     l_s.toDense(l);
912     u_s.toDense(u);
913     mul(l, u, a);
914     a.negate();
915     sum(m, a, sub);
916
917     for (int i = 0; i < sub.height(); ++i) {
918         for (size_t j = 0; j < sub.width(); ++j) {
919             if (fabs(sub(i, j)) < 0.000001)
920                 sub(i, j) = 0;
921         }
922     }
923
924     fout << "a in dense:" << endl;
925     m.save(fout);
926     fout << endl;
927
928     fout << "l in dense:" << endl;
929     l.save(fout);
930     fout << endl;
931
932     fout << "u in dense:" << endl;
933     u.save(fout);
934     fout << endl;
935
936     fout << "a - l*u:" << endl;
937     sub.save(fout);
938     fout << endl;
939
940     pr = to(s.f);
941     f1 = s.f;
942     mul(m, pr, pr1);
943     mul(s.a, f1);
944     fout << "a * vec:" << endl;
945     fout << to(pr1) << endl;
946     fout << f1 << endl;
947     fout << endl;
948 }

```

```

965 pr = to(s.f);
966 f1 = s.f;
967 transpose(m);
968 mul(m, pr, pr1);
969 transpose(m);
970 mul_t(s.a, f1);
971 fout << "a * vec:" << endl;
972 fout << to(pr1) << endl;
973 fout << f1 << endl;
974 fout << endl;
975 pr = to(s.f);
976 f1 = s.f;
977 mul(u, pr, pr1);
978 mul_u(s.lu, f1);
979 fout << "u * vec:" << endl;
980 fout << to(pr1) << endl;
981 fout << f1 << endl;
982 fout << endl;
983 pr = to(s.f);
984 mul(l, pr, pr1);
985 f1 = to(pr1);
986 pr1 = pr;
987 mul_l_invert(s.lu, f1);
988 fout << "l^-1 * vec:" << endl;
989 fout << to(pr1) << endl;
990 fout << f1 << endl;
991 fout << endl;
992 pr = to(s.f);
993 transpose(u);
994 mul(u, pr, pr1);
995 transpose(u);
996 f1 = to(pr1);
997 pr1 = pr;
998 mul_u_invert_t(s.lu, f1);
999 fout << "u^-t * vec:" << endl;
1000 fout << to(pr1) << endl;
1001 fout << f1 << endl;
1002 fout << endl;
1003 pr = to(s.f);
1004 mul(u, pr, pr1);
1005 f1 = to(pr1);
1006 pr1 = pr;
1007 mul_u_invert(s.lu, f1);
1008 fout << "u^-1 * vec:" << endl;
1009 fout << to(pr1) << endl;
1010 fout << f1 << endl;
1011 fout << endl;
1012 pr = to(s.f);
1013 transpose(l);
1014 mul(l, pr, pr1);
1015 transpose(l);
1016 f1 = to(pr1);
1017 pr1 = pr;
1018 mul_l_invert_t(s.lu, f1);
1019 fout << "l^-t * vec:" << endl;
1020 fout << to(pr1) << endl;
1021 fout << f1 << endl;
1022 fout << endl;
1023 pr = to(s.f);
1024 transpose(l);
1025 mul(l, pr, pr1);
1026 transpose(l);
1027 f1 = to(pr1);
1028 pr1 = pr;
1029 mul_l_invert(s.lu, f1);
1030 fout << "l^-1 * vec:" << endl;
1031 fout << to(pr1) << endl;
1032 fout << f1 << endl;
1033 fout << endl;
1034 }
1035 }
1036
1037 //-----
1038 string print_time(double time) {
1039     stringstream sout;
1040     if (time > 1000 * 1000) {
1041         sout << time / (1000 * 1000) << " s";
1042     } else if (time > 1000) {
1043         sout << time / 1000 << " ms";
1044     } else {
1045         sout << time << " us";
1046     }
1047     return sout.str();
1048 }
1049
1050 void test_method(string name, function<pair<int, double>(SLAU)> f, SLAU& s, bool
1051 → is_write_each_iteration) {
1052     cout << name << ":" << endl;
1053
1054     s.is_log = false;
1055     int count = 100;
1056     double time = 0;
1057     pair<int, double> temp_res;
1058     chrono::high_resolution_clock::time_point t1, t2;
1059     for (int i = 0; i < count; i++) {
1060         t1 = chrono::high_resolution_clock::now();
1061         temp_res = f(s);
1062         t2 = chrono::high_resolution_clock::now();
1063         time += chrono::duration_cast<chrono::microseconds>(t2 - t1).count();
1064         if (time / 1000.0 > 1000.0) count = i+1;
1065     }
1066     time /= count;
1067
1068     if (is_write_each_iteration) {
1069         s.is_log = true;
1070         f(s);
1071     } else {
1072         cout << "Iterations: " << temp_res.first << ", Residual: " << temp_res.second << endl;
1073     }
1074     cout << "Time: " << print_time(time) << endl;
1075     cout << "X: " << s.x << endl << endl;
1076 }
1077
1078 //-----
1079
1080 int main() {
1081     //for (int i = 0; i < 16; ++i) make_gilbert(i);
1082
1083     string dir = "test1";
1084     bool is_write_to_file = true;
1085     bool is_write_each_iteration = false;
1086     bool is_write_tests = false;
1087
1088     cout << "Enter dir: ";
1089     cin >> dir;
1090     cout << "Is write to file? (0 or 1): ";
1091     cin >> is_write_to_file;
1092     cout << "Is write each iteration? (0 or 1): ";
1093     cin >> is_write_each_iteration;
1094     cout << "Is write tests? (0 or 1): ";
1095     cin >> is_write_tests;
1096
1097     if (is_write_to_file) freopen((dir + "/res.txt").c_str(), "w", stdout);
1098
1099     SLAU s;
1100     s.read(dir);
1101
1102     if (is_write_tests) test(s, dir);
1103
1104     test_method("MSG", &SLAU::msg1, s, is_write_each_iteration);
1105     test_method("MSG [Usq]", &SLAU::msg2, s, is_write_each_iteration);
1106     test_method("MSG D", &SLAU::msg3, s, is_write_each_iteration);
1107     test_method("LOS", &SLAU::los1, s, is_write_each_iteration);
1108     test_method("LOS [Usq]", &SLAU::los2, s, is_write_each_iteration);
1109     test_method("LOS D", &SLAU::los3, s, is_write_each_iteration);
1110
1111     if (!is_write_to_file) system("pause");
1112 }

```