

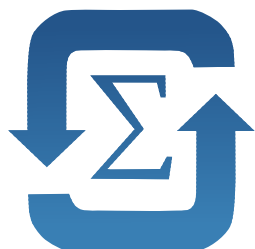
Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



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

Метод	Итераций	Действий за итерацию	Всего действий
Якоби	132	$2n^2 + 4n$	$264n^2 + 528n$
Гаусс-Зейдель	57	$2n^2 + 4n$	$114n^2 + 228n$
МСГ LU(sq)	10	$8n^2 + 12n$	$80n^2 + 120n$
ЛОС	119	$2n^2 + 14n$	$238n^2 + 1666n$
ЛОС LU(sq)	26	$5n^2 + 14n$	$130n^2 + 364n$
ЛОС Диаг.	88	$2n^2 + 17n$	$176n^2 + 1496n$

Эта же таблица в пересчете, что каждая матрица имеет 7 диагоналей:

Метод	Итераций	Действий за итерацию	Всего действий
Якоби	132	$18n$	$2376n$
Гаусс-Зейдель	57	$18n$	$1026n$
МСГ LU(sq)	10	$72n$	$720n$
ЛОС	119	$28n$	$3332n$
ЛОС LU(sq)	26	$52n$	$1352n$
ЛОС Диаг.	88	$31n$	$2728n$

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 предобуславливания, 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 //ostream& operator<<(ostream& out, const vector<double>& x) {
17 out.precision(16);
18 out << "x=" << endl;
19 if (x.size() != 0) {
20     for (int i = 0; i < x.size() - 1; ++i)
21         out << x[i] << " ";
22     out << x.back() << " ";
23 } else {
24     out << "x=" << endl;
25 }
26 return out;
27 }
28
29 //double operator*(const vector<double>& a, const vector<double>& b) {
30 double sum = 0;
31 for (int i = 0; i < a.size(); ++i)
32     sum += a[i] * b[i];
33 return sum;
34 }
35
36 //double length(const vector<double>& mas) {
37 return sqrt(mas*mas);
38 }
39
40 //vector<double> to(const Vector& a) {
41 vector<double> result(a.size());
42 for (int i = 0; i < a.size(); ++i)
43     result[i] = a[i];
44 return result;
45 }
46
47 //-----
48
49 Vector to(const vector<double>& a) {
50     Vector result(a.size());
51     for (int i = 0; i < a.size(); ++i)
52         result[i] = a[i];
53     return result;
54 }
55
56 //-----
57
58 //-----
59
60 //-----
61
62 //-----
63
64 //-----
65
66 //-----
67
68 //-----
69
70 //-----
71
72 //-----
73
74 //-----
75
76 //-----
77
78 //-----
79
80 //-----
81
82 //-----
83
84 //-----
85
86 //-----
87
88 //-----
89
90 //-----
91
92 //-----
93
94 //-----
95
96 //-----
97
98 //-----
99
100 //-----
101
102 //-----
103
104 //-----
105
106 //-----
107
108 //-----
109
110 //-----
111
112 //-----
113
114 //-----
115
116 //-----
117
118 //-----
119
120 //-----
121
122 //-----
123
124 //-----
125
126 //-----
127
128 //-----
129
130 //-----
131
132 //-----
133
134 //-----
135
136 //-----
137
138 //-----
139
140 //-----
141
142 //-----
143
144 //-----
145
146 //-----
147
148 //-----
149
150 //-----
151
152 //-----
153
154 //-----
155
156 //-----
157
158 //-----
159
160 //-----
161
162 //-----
163
164 //-----
165
166 //-----
167
168 //-----
169
170 //-----
171
172 //-----
173
174 //-----
175
176 //-----
177
178 //-----
179
180 //-----
181
182 //-----
183
184 //-----
185
186 //-----
187
188 //-----
189
190 //-----
191
192 //-----
193
194 //-----
195
196 //-----
197
198 //-----
199
200 //-----
201
202 //-----
203
204 //-----
205
206 //-----
207
208 //-----
209
210 //-----
211
212 //-----
213
214 //-----
215
216 //-----
217
218 //-----
219
220 //-----
221
222 //-----
223
224 //-----
225
226 //-----
227
228 //-----
229
230 //-----
231
232 //-----
233
234 //-----
235
236 //-----
237
238 //-----
239
240 //-----
241
242 //-----
243
244 //-----
245
246 //-----
247
248 //-----
249
250 //-----
251
252 //-----
253
254 //-----
255
256 //-----
257
258 //-----
259
260 //-----
261
262 //-----
263
264 //-----
265
266 //-----
267
268 //-----
269
270 //-----
271
272 //-----
273
274 //-----
275
276 //-----
277
278 //-----
279
280 //-----
281
282 //-----
283
284 //-----
285
286 //-----
287
288 //-----
289
290 //-----
291
292 //-----
293
294 //-----
295
296 //-----
297
298 //-----
299
300 //-----
301
302 //-----
303
304 //-----
305
306 //-----
307
308 //-----
309
310 //-----
311
312 //-----
313
314 //-----
315
316 //-----
317
318 //-----
319
320 //-----
321
322 //-----
323
324 //-----
325
326 //-----
327
328 //-----
329
330 //-----
331
332 //-----
333
334 //-----
335
336 //-----
337
338 //-----
339
340 //-----
341
342 //-----
343
344 //-----
345
346 //-----
347
348 //-----
349
350 //-----
351
352 //-----
353
354 //-----
355
356 //-----
357
358 //-----
359
360 //-----
361
362 //-----
363
364 //-----
365
366 //-----
367
368 //-----
369
370 //-----
371
372 //-----
373
374 //-----
375
376 //-----
377
378 //-----
379
380 //-----
381
382 //-----
383
384 //-----
385
386 //-----
387
388 //-----
389
390 //-----
391
392 //-----
393
394 //-----
395
396 //-----
397
398 //-----
399
400 //-----
401
402 //-----
403
404 //-----
405
406 //-----
407
408 //-----
409
410 //-----
411
412 //-----
413
414 //-----
415
416 //-----
417
418 //-----
419
420 //-----
421
422 //-----
423
424 //-----
425
426 //-----
427
428 //-----
429
430 //-----
431
432 //-----
433
434 //-----
435
436 //-----
437
438 //-----
439
440 //-----
441
442 //-----
443
444 //-----
445
446 //-----
447
448 //-----
449
450 //-----
451
452 //-----
453
454 //-----
455
456 //-----
457
458 //-----
459
460 //-----
461
462 //-----
463
464 //-----
465
466 //-----
467
468 //-----
469
470 //-----
471
472 //-----
473
474 //-----
475
476 //-----
477
478 //-----
479
480 //-----
481
482 //-----
483
484 //-----
485
486 //-----
487
488 //-----
489
490 //-----
491
492 //-----
493
494 //-----
495
496 //-----
497
498 //-----
499
500 //-----
501
502 //-----
503
504 //-----
505
506 //-----
507
508 //-----
509
510 //-----
511
512 //-----
513
514 //-----
515
516 //-----
517
518 //-----
519
520 //-----
521
522 //-----
523
524 //-----
525
526 //-----
527
528 //-----
529
530 //-----
531
532 //-----
533
534 //-----
535
536 //-----
537
538 //-----
539
540 //-----
541
542 //-----
543
544 //-----
545
546 //-----
547
548 //-----
549
550 //-----
551
552 //-----
553
554 //-----
555
556 //-----
557
558 //-----
559
560 //-----
561
562 //-----
563
564 //-----
565
566 //-----
567
568 //-----
569
570 //-----
571
572 //-----
573
574 //-----
575
576 //-----
577
578 //-----
579
580 //-----
581
582 //-----
583
584 //-----
585
586 //-----
587
588 //-----
589
590 //-----
591
592 //-----
593
594 //-----
595
596 //-----
597
598 //-----
599
600 //-----
601
602 //-----
603
604 //-----
605
606 //-----
607
608 //-----
609
610 //-----
611
612 //-----
613
614 //-----
615
616 //-----
617
618 //-----
619
620 //-----
621
622 //-----
623
624 //-----
625
626 //-----
627
628 //-----
629
630 //-----
631
632 //-----
633
634 //-----
635
636 //-----
637
638 //-----
639
640 //-----
641
642 //-----
643
644 //-----
645
646 //-----
647
648 //-----
649
650 //-----
651
652 //-----
653
654 //-----
655
656 //-----
657
658 //-----
659
660 //-----
661
662 //-----
663
664 //-----
665
666 //-----
667
668 //-----
669
670 //-----
671
672 //-----
673
674 //-----
675
676 //-----
677
678 //-----
679
680 //-----
681
682 //-----
683
684 //-----
685
686 //-----
687
688 //-----
689
690 //-----
691
692 //-----
693
694 //-----
695
696 //-----
697
698 //-----
699
700 //-----
701
702 //-----
703
704 //-----
705
706 //-----
707
708 //-----
709
710 //-----
711
712 //-----
713
714 //-----
715
716 //-----
717
718 //-----
719
720 //-----
721
722 //-----
723
724 //-----
725
726 //-----
727
728 //-----
729
730 //-----
731
732 //-----
733
734 //-----
735
736 //-----
737
738 //-----
739
740 //-----
741
742 //-----
743
744 //-----
745
746 //-----
747
748 //-----
749
750 //-----
751
752 //-----
753
754 //-----
755
756 //-----
757
758 //-----
759
760 //-----
761
762 //-----
763
764 //-----
765
766 //-----
767
768 //-----
769
770 //-----
771
772 //-----
773
774 //-----
775
776 //-----
777
778 //-----
779
780 //-----
781
782 //-----
783
784 //-----
785
786 //-----
787
788 //-----
789
790 //-----
791
792 //-----
793
794 //-----
795
796 //-----
797
798 //-----
799
800 //-----
801
802 //-----
803
804 //-----
805
806 //-----
807
808 //-----
809
810 //-----
811
812 //-----
813
814 //-----
815
816 //-----
817
818 //-----
819
820 //-----
821
822 //-----
823
824 //-----
825
826 //-----
827
828 //-----
829
830 //-----
831
832 //-----
833
834 //-----
835
836 //-----
837
838 //-----
839
840 //-----
841
842 //-----
843
844 //-----
845
846 //-----
847
848 //-----
849
850 //-----
851
852 //-----
853
854 //-----
855
856 //-----
857
858 //-----
859
860 //-----
861
862 //-----
863
864 //-----
865
866 //-----
867
868 //-----
869
870 //-----
871
872 //-----
873
874 //-----
875
876 //-----
877
878 //-----
879
880 //-----
881
882 //-----
883
884 //-----
885
886 //-----
887
888 //-----
889
890 //-----
891
892 //-----
893
894 //-----
895
896 //-----
897
898 //-----
899
900 //-----
901
902 //-----
903
904 //-----
905
906 //-----
907
908 //-----
909
910 //-----
911
912 //-----
913
914 //-----
915
916 //-----
917
918 //-----
919
920 //-----
921
922 //-----
923
924 //-----
925
926 //-----
927
928 //-----
929
930 //-----
931
932 //-----
933
934 //-----
935
936 //-----
937
938 //-----
939
940 //-----
941
942 //-----
943
944 //-----
945
946 //-----
947
948 //-----
949
950 //-----
951
952 //-----
953
954 //-----
955
956 //-----
957
958 //-----
959
960 //-----
961
962 //-----
963
964 //-----
965
966 //-----
967
968 //-----
969
970 //-----
971
972 //-----
973
974 //-----
975
976 //-----
977
978 //-----
979
980 //-----
981
982 //-----
983
984 //-----
985
986 //-----
987
988 //-----
989
990 //-----
991
992 //-----
993
994 //-----
995
996 //-----
997
998 //-----
999
1000 //-----
```

```

110 //-----
111 //-----
112 //-----
113 #ifndef FULL_FACTORIZATION
114 struct matrix_iterator
115 {
116     matrix_iterator(const matrix& m, int line) : m(m), line(line) {
117         if (m.lineElemCount(line) != 0) {
118             pos = m.lineStart(line);
119             j = m.lineElemStart(line);
120             is_on_elem = true;
121             is_empty_line = false;
122             line_size = m.lineElemCount(line);
123         } else {
124             pos = 0;
125             j = 0;
126             is_on_elem = false;
127             is_empty_line = true;
128             line_size = 0;
129         }
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 int getpos(void) const { return pos; }
145 protected:
146 bool is_on_elem, is_empty_line;
147 const matrix& m;
148 int line, line_size;
149 int pos, j, end;
150 };
151 struct matrix_iterator_l : public matrix_iterator
152 {
153     matrix_iterator_l(const matrix& m, int line) : matrix_iterator(m, line) {}
154     double operator*() const {
155         if (is_on_elem) return m.l[j];
156         else return 0;
157     }
158 };
159 struct matrix_iterator_u : public matrix_iterator
160 {
161     matrix_iterator_u(const matrix& m, int row) : matrix_iterator(m, row) {}
162     double operator*() const {
163         if (is_on_elem) return m.u[j];
164         else return 0;
165     }
166 };
167 struct line_iterator
168 {
169     line_iterator(const vector<pair<int, double>>& line) : line(line), i(0), pos(0),
170     ↪ is_on_elem(line.size() != 0) {
171         if (line.size() != 0) {
172             pos = line[0].first;
173             is_on_elem = true;
174         }
175     }
176     line_iterator& operator++() {
177         if (line.size() != 0 && pos+1 <= line.back().first && i + 1 < line.size() && pos+1 ==
178             ↪ line[i+1].first) {
179             is_on_elem = true;
180             pos++;
181             i++;
182         } else {
183             is_on_elem = false;
184             pos++;
185         }
186         return *this;
187     }
188     double operator*() const {
189         if (is_on_elem) return line[i].second;
190         else return 0;
191     }
192     int getpos(void) const { return pos; }
193 protected:
194 const vector<pair<int, double>>& line;
195 int i, pos;
196 bool is_on_elem;
197 };
198 template<class T1, class T2>
199 void synchronize_iterators(T1& i, T2& j) {
200     while (i.getpos() != j.getpos()) {
201         if (i.getpos() < j.getpos())
202             ++i;
203         else
204             ++j;
205     }
206 }
207 #endif
208 void lu_decompose(const matrix& a, matrix& lu) {
209     #ifndef FULL_FACTORIZATION
210     lu.init(a.n);
211     for (int i = 0; i < a.n; ++i) {
212         // Считаем элементы матрицы L
213         vector<pair<int, double>> l_add, u_add;
214         {
215             matrix_iterator_l a_j(a, i);
216             int iline_start = a.lineStart(i);
217             for (int j = iline_start; j < i; ++j, ++a_j) {
218                 double sum = 0;
219                 if (j != iline_start) {
220                     line_iterator_l k_l(a, j);
221                     matrix_iterator_u u_k(u, k_l.u, j);
222                     synchronize_iterators(l_k, u_k);
223                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
224                         sum += (*l_k) * (*u_k);
225                 }
226                 double res = ((*a_j) - sum) / lu.d[j];
227                 if (res != 0)
228                     l_add.push_back({j, res});
229             }
230         }
231         // Считаем элементы матрицы U
232         {
233             matrix_iterator_u a_j(a, i);
234             int irow_start = a.lineStart(i);
235             for (int j = irow_start; j < i; ++j, ++a_j) {
236                 double sum = 0;
237                 if (j != irow_start) {
238                     matrix_iterator_l l_k(lu, j);
239                     line_iterator u_k(u, a_j.u, j);
240                     synchronize_iterators(l_k, u_k);
241                     for (int k = l_k.getpos(); k < j; ++k, ++l_k, ++u_k)
242                         sum += (*l_k) * (*u_k);
243                 }
244                 double res = ((*a_j) - sum) / lu.d[j];
245                 if (res != 0)
246                     u_add.push_back({j, res});
247             }
248         }
249     }
250     // Находим такие элементы, которые имеются в одном массиве, но нет в другом
251     int j = 0;
252     while (j < l_add.size() || j < u_add.size()) {
253         if (j >= u_add.size())
254             u_add.insert(u_add.begin() + j, {l_add[j].first, 0});
255         else
256             l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
257         if (l_add[j].first != u_add[j].first) {
258             if (u_add[j].first > j, l_add[j].first, 0);
259             else
260                 l_add.insert(l_add.begin() + j, {u_add[j].first, 0});
261         }
262         j++;
263     }
264     if (l_add.size() != u_add.size())
265         throw std::exception();
266 }

```

```

284 // Добавляем формат к обоим массивам
285 lu.i[i+1] = lu.i[i] + l_add.size();
286 for (int j = 0; j < l_add.size(); ++j) {
287     lu.j.push_back(l_add[j].first);
288     lu.l.push_back(l_add[j].second);
289     lu.u.push_back(u_add[j].second);
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 #endif
301 #ifndef FULL_FACTORIZATION
302 lu = a;
303 for (int i = 0; i < lu.n; ++i) {
304     // Заполняем нижний треугольник
305     int line_start = lu.lineElemStart(i);
306     int line_end = lu.lineElemStart(i+1);
307     for (int j = line_start; j < line_end; ++j) {
308         double sum = 0;
309         int row = lu.j[j];
310         int row_start = lu.lineElemStart(row);
311         int row_end = lu.lineElemStart(row+1);
312         int kl = line_start;
313         int ku = row_start;
314         while (kl < j && ku < row_end) {
315             if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
316                 sum += lu.l[kl] * lu.u[ku];
317                 ku++;
318             } else if (lu.j[kl] < lu.j[ku]) {
319                 kl++;
320             } else {
321                 ku++;
322             }
323         }
324         lu.l[j] = (a.l[j] - sum) / lu.d[row];
325     }
326     // Заполняем верхний треугольник
327     int row_start = lu.lineElemStart(i);
328     int row_end = lu.lineElemStart(i+1);
329     for (int j = line_start; j < line_end; ++j) {
330         double sum = 0;
331         int line = lu.j[j];
332         int line_start = lu.lineElemStart(line);
333         int line_end = lu.lineElemStart(line+1);
334         int kl = line_start;
335         int ku = row_start;
336         while (kl < line_end && ku < j) {
337             if (lu.j[kl] == lu.j[ku]) { // Совпадают столбцы
338                 sum += lu.l[kl] * lu.u[ku];
339                 ku++;
340             } else if (lu.j[kl] < lu.j[ku]) {
341                 kl++;
342             } else {
343                 ku++;
344             }
345         }
346         lu.u[j] = (a.u[j] - sum) / lu.d[line];
347     }
348     // Рассчитываем диагональный элемент
349     double sum = 0;
350     int line_row_start = lu.lineElemStart(i);
351     int line_row_end = lu.lineElemStart(i+1);
352     for (int j = line_row_start; j < line_row_end; ++j)
353         sum += lu.l[j] * lu.u[j];
354     lu.d[i] = sqrt(a.d[i] - sum);
355 }
356 #endif
357 //-----
358 //-----
359 //-----
360 void mul(const matrix& a, vector<double>& x_y) {
361     vector<double> result(a.n, 0);
362     for (int i = 0; i < a.n; ++i) {
363         int start = a.lineElemStart(i);
364         int size = a.lineElemCount(i);
365         for (int j = 0; j < size; ++j) {
366             result[i] += a.l[start + j] * x_y[a.lineElemRow(i, j)];
367             result[a.lineElemRow(i, j)] += a.u[start + j] * x_y[i];
368         }
369     }
370     // Умножение диагональных элементов на вектор
371     for (int i = 0; i < a.n; ++i)
372         result[i] += a.d[i] * x_y[i];
373     x_y = result;
374 }
375 //-----
376 void mul_t(const matrix& a, vector<double>& x_y) {
377     vector<double> result(a.n, 0);
378     for (int i = 0; i < a.n; ++i) {
379         int start = a.lineElemStart(i);
380         int size = a.lineElemCount(i);
381         for (int j = 0; j < size; ++j) {
382             result[i] += a.u[start + j] * x_y[a.lineElemRow(i, j)];
383             result[a.lineElemRow(i, j)] += a.l[start + j] * x_y[i];
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     x_y = result;
390 }
391 //-----
392 void mul_l_invert_t(const matrix& l, vector<double>& y_x) {
393     for (int i = l.n; i >= 0; i--) {
394         int start = l.lineElemStart(i);
395         int size = l.lineElemCount(i);
396         y_x[i] /= l.d[i];
397         for (int j = 0; j < size; ++j)
398             y_x[l.lineElemRow(i, j)] -= y_x[i] * l.l[start + j];
399     }
400 }
401 //-----
402 void mul_u_invert_t(const matrix& u, vector<double>& y_x) {
403     for (int i = 0; i < u.n; ++i) {
404         int start = u.lineElemStart(i);
405         int size = u.lineElemCount(i);
406         sumreal sum = 0;
407         for (int j = 0; j < size; ++j)
408             sum += u.u[start + j] * y_x[u.lineElemRow(i, j)];
409         y_x[i] = (y_x[i] - sum) / u.d[i];
410     }
411 }
412 //-----
413 void mul_l_invert(const matrix& l, vector<double>& y_x) {
414     for (int i = l.n; i >= 0; i--) {
415         int start = l.lineElemStart(i);
416         int size = l.lineElemCount(i);
417         sumreal sum = 0;
418         for (int j = 0; j < size; ++j)
419             sum += l.l[start + j] * y_x[l.lineElemRow(i, j)];
420         y_x[i] = (y_x[i] - sum) / l.d[i];
421     }
422 }
423 //-----
424 void mul_u_invert(const matrix& u, vector<double>& y_x) {
425     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 void mul_u(const matrix& u, vector<double>& x_y) {
472     vector<double> result(u.n, 0);
473
474     for (int i = 0; i < u.n; ++i) {
475         int start = u.lineElemStart(i);
476         int size = u.lineElemCount(i);
477         for (int j = 0; j < size; ++j) {
478             result[u.lineElemRow(i, j)] += u.u[start + j] * x_y[j];
479         }
480     }
481
482     // Умножение диагональных элементов на вектор
483     for (int i = 0; i < u.n; ++i)
484         result[i] += u.d[i] * x_y[i];
485
486     x_y = result;
487 }
488
489 //-----
490 void mul(const vector<double>& d, vector<double>& x_y) {
491     for (int i = 0; i < d.size(); ++i)
492         x_y[i] *= d[i];
493 }
494
495 //-----
496 void mul_invert(const vector<double>& d, vector<double>& x_y) {
497     for (int i = 0; i < d.size(); ++i)
498         x_y[i] /= d[i];
499 }
500
501 //-----
502 //-----
503 //-----
504 //-----
505 //-----
506 //-----
507 class SLAU
508 {
509 public:
510
511     //-----
512     void read(string dir) {
513         ifstream fin;
514
515         fin.open(dir + "/kuslau.txt");
516         fin >> n >> maxiter >> eps;
517         fin.close();
518
519         a.n = n;
520
521         a.d.resize(n);
522         fin.open(dir + "/di.txt");
523         for (auto& i : a.d) fin >> i;
524         fin.close();
525
526         f.resize(n);
527         fin.open(dir + "/pr.txt");
528         for (auto& i : f) fin >> i;
529         fin.close();
530
531         a.i.resize(n+1);
532         fin.open(dir + "/ig.txt");
533         for (auto& i : a.i) { fin >> i; i--; }
534         fin.close();
535
536         a.j.resize(a.i.back());
537         fin.open(dir + "/jg.txt");
538         for (auto& i : a.j) { fin >> i; i--; }
539         fin.close();
540
541         a.l.resize(a.i.back());
542         fin.open(dir + "/lgl.txt");
543         for (auto& i : a.l) { fin >> i; }
544         fin.close();
545
546         a.u.resize(a.i.back());
547         fin.open(dir + "/ugl.txt");
548         for (auto& i : a.u) { fin >> i; }
549         fin.close();
550
551         x.resize(n);
552         t1.resize(n);
553         t2.resize(n);
554
555         is_log = true;
556     }
557
558     //-----
559     pair<int, double> msg1() {
560         x.clear();
561         x.resize(n, 0);
562
563         r = x;
564         mul(a, r);
565         for (int i = 0; i < n; ++i)
566             r[i] = f[i] - r[i];
567
568         z = r;
569         double rr = r*r;
570         double flen = sqrt(f*f);
571         double residual;
572
573         int i = 0;
574         while (true) {
575             t1 = z;
576             mul(a, t1);
577             double alpha = (rr) / (t1*t1);
578             for (int i = 0; i < n; ++i) {
579                 x[i] += alpha * z[i];
580                 r[i] -= alpha * t1[i];
581             }
582             double rr2 = r*r;
583             double beta = rr2/rr;
584             rr = rr2;
585             for (int i = 0; i < n; ++i)
586                 z[i] = r[i] + beta * z[i];
587             residual = sqrt(rr) / flen;
588             i++;
589
590             if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
591                 << setprecision(16) << residual << endl;
592             if (fabs(residual) < eps || i > maxiter)
593                 break;
594         }
595         return {i, residual};
596     }
597
598     //-----
599     pair<int, double> msg2() {
600         lu_decompose(a, lu);
601
602         x.clear();
603         x.resize(n, 0);
604
605         r = x;
606         mul(a, r);
607         for (int i = 0; i < n; ++i)
608             r[i] = f[i] - r[i];
609         mul_l_invert(lu, r);
610         mul_l_invert_t(lu, r);
611         mul_t(a, r);
612         mul_u_invert_t(lu, r);
613
614         mul_u(lu, x);
615
616         z = r;
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_l_invert(lu, t1);
626             mul_l_invert_t(lu, t1);
627             mul_t(a, t1);
628             mul_u_invert_t(lu, t1);
629             double alpha = (rr) / (t1*t1);
630             for (int i = 0; i < n; ++i) {
631                 x[i] += alpha * z[i];
632                 r[i] -= alpha * t1[i];
633             }
634             double rr2 = r*r;
635             double beta = rr2/rr;
636             rr = rr2;
637

```

```

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) <<
644             << setprecision(16) << residual << endl;
645         if (fabs(residual) < eps || i > maxiter)
646             break;
647     }
648
649     mul_u_invert(lu, x);
650
651     return {i, residual};
652 }
653
654 //-----
655 pair<int, double> msg3() {
656     x.clear();
657     x.resize(n, 0);
658
659     r = x;
660     mul(a, r);
661     for (int i = 0; i < n; ++i)
662         r[i] = f[i] - r[i];
663
664     z = r;
665     mul_invert(a.d, z);
666
667     double rr;
668     t1 = r;
669     mul_invert(a.d, t1);
670     rr = t1*t1;
671     double flen = sqrt(f*f);
672     double residual;
673
674     int i = 0;
675     while (true) {
676         t1 = z;
677         mul(a, t1);
678         double alpha = (rr) / (t1*t1);
679         for (int i = 0; i < n; ++i) {
680             x[i] += alpha * z[i];
681             r[i] -= alpha * t1[i];
682         }
683         t1 = r;
684         mul_invert(a.d, t1);
685         double rr2 = t1*t1;
686         double beta = rr2/rr;
687         rr = rr2;
688         for (int i = 0; i < n; ++i)
689             z[i] = t1[i] + beta * z[i];
690         residual = length(r) / flen;
691         i++;
692
693         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
694             << setprecision(16) << residual << endl;
695         if (fabs(residual) < eps || i > maxiter)
696             break;
697     }
698     return {i, residual};
699 }
700
701 //-----
702 pair<int, double> los1() {
703     x.clear();
704     x.resize(n, 0);
705
706     r = x;
707     mul(a, r);
708     for (int i = 0; i < n; ++i)
709         r[i] = f[i] - r[i];
710
711     z = r;
712     p = z;
713     mul(a, p);
714
715     double flen = sqrt(f*f);
716     double residual;
717
718     int i = 0;
719     while (true) {
720         double pp = p*p;
721         double alpha = (p*r) / pp;
722         for (int i = 0; i < n; ++i) {
723             x[i] += alpha * z[i];
724             r[i] -= alpha * p[i];
725         }
726         t1 = r;
727         mul(a, t1);
728         double beta = -(p*t1) / pp;
729         for (int i = 0; i < n; ++i) {
730             z[i] = r[i] + beta * z[i];
731             p[i] = t1[i] + beta * p[i];
732         }
733         residual = length(r) / flen;
734         i++;
735
736         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
737             << setprecision(16) << residual << endl;
738         if (fabs(residual) < eps || i > maxiter)
739             break;
740     }
741     return {i, residual};
742 }
743
744 //-----
745 pair<int, double> los2() {
746     lu_decompose(a, lu);
747     x.clear();
748     x.resize(n, 0);
749
750     r = x;
751     mul(a, r);
752     for (int i = 0; i < n; ++i)
753         r[i] = f[i] - r[i];
754     mul_l_invert(lu, r);
755
756     z = r;
757     mul_u_invert(lu, z);
758
759     p = z;
760     mul_l_invert(lu, p);
761     mul_l_invert_t(lu, p);
762
763     double flen = sqrt(f*f);
764     double residual;
765
766     int i = 0;
767     while (true) {
768         double pp = p*p;
769         double alpha = (p*r) / pp;
770         for (int i = 0; i < n; ++i) {
771             x[i] += alpha * z[i];
772             r[i] -= alpha * p[i];
773         }
774         t1 = r;
775         mul_u_invert(lu, t1);
776         t2 = t1;
777         mul(a, t2);
778         mul_l_invert(lu, t2);
779         double beta = -(p*t2) / pp;
780         for (int i = 0; i < n; ++i) {
781             z[i] = t1[i] + beta * z[i];
782             p[i] = t2[i] + beta * p[i];
783         }
784         residual = length(r) / flen;
785         i++;
786
787         if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
788             << setprecision(16) << residual << endl;
789         if (fabs(residual) < eps || i > maxiter)
790             break;
791     }
792     return {i, residual};
793 }
794
795 //-----
796 pair<int, double> los3() {
797     x.clear();
798     x.resize(n, 0);
799
800     r = x;
801     mul(a, r);
802     for (int i = 0; i < n; ++i)
803         r[i] = f[i] - r[i];
804     mul_invert(a.d, r);
805
806     z = r;
807     mul_invert(a.d, z);
808
809     p = z;

```

```

810 mul(a, p);
811 mul_invert(a.d, p);
812
813 double flen = sqrt(f*f);
814 double residual;
815
816 int i = 0;
817 while (true) {
818     double pp = p*p;
819     double alpha = (p*r) / pp;
820     for (int i = 0; i < n; ++i) {
821         x[i] += alpha * z[i];
822         r[i] -= alpha * p[i];
823     }
824     t1 = r;
825     mul_invert(a.d, t1);
826     t2 = t1;
827     mul(a, t2);
828     mul_invert(a.d, t2);
829     double beta = -(p*t2) / pp;
830     for (int i = 0; i < n; ++i) {
831         z[i] = t1[i] + beta * z[i];
832         p[i] = t2[i] + beta * p[i];
833     }
834     residual = length(r) / flen;
835     i++;
836     if (is_log) cout << "Iteration: " << setw(4) << i << ", Residual: " << setw(20) <<
837         << setprecision(16) << residual << endl;
838     if (fabs(residual) < eps || i > maxiter)
839         break;
840 }
841 return {i, residual};
842 }
843
844 int n, maxiter;
845 double eps;
846 matrix<double> a, lu;
847 vector<double> f;
848 vector<double> r, z, p;
849 vector<double> x, t1, t2;
850 bool is_log;
851
852 };
853
854 //-----
855 //-----
856 //-----
857 //-----
858 //-----
859
860 void make_gilbert(int size) {
861     Matrix g;
862     generateGilbertMatrix(size, g);
863     Vector x, y;
864     x.generate(size);
865     mul(g, x, y);
866
867     string dir = "gilbert" + to_string(size);
868     system(("mkdir " + dir).c_str());
869     ofstream fout;
870     fout.precision(16);
871
872     fout.open(dir + "/kusi1au.txt");
873     fout << size << " " << 1000 << " " << 1e-13;
874     fout.close();
875
876     fout.open(dir + "/d1.txt");
877     for (int i = 0; i < size; ++i)
878         fout << double(1.0)/double((i+1)*(i+1)-1) << " ";
879     fout.close();
880
881     fout.open(dir + "/pr.txt");
882     for (int i = 0; i < size; ++i)
883         fout << y[i] << " ";
884     fout.close();
885
886     fout.open(dir + "/ig.txt");
887     int sum = 1;
888     fout << sum << " ";
889     for (int i = 0; i < size; ++i) {
890         sum += i;
891         fout << sum << " ";
892     }
893     fout.close();
894
895     fout.open(dir + "/jg.txt");
896     for (int i = 0; i < size; ++i)
897         for (int j = 0; j < i; ++j)
898             fout << j+1 << " ";
899     fout.close();
900
901     fout.open(dir + "/ggl.txt");
902     for (int i = 0; i < size; ++i)
903         for (int j = 0; j < i; ++j)
904             fout << double(1.0)/double((i+1)*(j+1)-1) << " ";
905     fout.close();
906
907     fout.open(dir + "/ggv.txt");
908     for (int i = 0; i < size; ++i)
909         for (int j = 0; j < i; ++j)
910             fout << double(1.0)/double((i+1)*(j+1)-1) << " ";
911     fout.close();
912 }
913
914 //-----
915 void test(SLAU& s, string dir) {
916     ofstream fout(dir + "test.txt");
917     vector<double> f1;
918     Matrix m, l, u, a, sub;
919     Vector pr(s.f.size()), pr1, f1_m(f1.size());
920     pr = to(f1);
921     s.a.toDense(m);
922
923     lu_decompose(s.a, s.lu);
924     matrix l_s = s.lu, u_s = s.lu;
925     l_s.u.clear(); l_s.u.resize(l_s.l.size(), 0);
926     u_s.l.clear(); u_s.l.resize(u_s.u.size(), 0);
927     l_s.toDense(l);
928     u_s.toDense(u);
929     mul(l, u, a);
930     a.negate();
931     sum(m, a, sub);
932
933     for (int i = 0; i < sub.height(); i++) {
934         for (size_t j = 0; j < sub.width(); j++) {
935             if (fabs(sub(i, j)) < 0.000001)
936                 sub(i, j) = 0;
937         }
938     }
939
940     fout << "a in dense:" << endl;
941     m.save(fout);
942     fout << endl;
943
944     fout << "l in dense:" << endl;
945     l.save(fout);
946     fout << endl;
947
948     fout << "u in dense:" << endl;
949     u.save(fout);
950     fout << endl;
951
952     fout << "a - l*u:" << endl;
953     sub.save(fout);
954     fout << endl;
955
956     pr = to(s.f);
957     f1 = s.f;
958     mul(m, pr, pr1);
959     mul(s.a, f1);
960     fout << "a * vec:" << endl;
961     fout << to(pr1) << endl;
962     fout << f1 << endl;
963     fout << endl;
964
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^T * vec:" << endl;
972     fout << to(pr1) << endl;
973     fout << f1 << endl;
974     fout << endl;
975
976     pr = to(s.f);
977     f1 = s.f;
978     mul(u, pr, pr1);
979     mul(u(s.lu, f1);
980     fout << "u * vec:" << endl;
981     fout << to(pr1) << endl;
982     fout << f1 << endl;
983     fout << endl;
984
985     pr = to(s.f);

```