Национальный исследовательский университет ИТМО
Прикладная математика и информатика

Методы оптимизации

Отчет по лабораторной работе №3
“Решение СЛАУ”

Выполнили:

Михайлов Максим
Загребина Мария
Кулагин Ярослав

Команда:
$\forall \bar{R} \in \mathscr{R}^n : \mathbf{R}(\bar{R}) \in \mathscr{R}$
(КаМаЗ)

Группа: M3237

Санкт-Петербург, 5 июня 2021 г.

1 Цель

1. Реализовать прямой метод решения СЛАУ на основе LU-разложения
2. Оценить влияние увеличения числа обусловленности на точность решения
3. Провести исследования на матрицах Гильберта различной размерности
4. Реализовать метод Гаусса с выбором ведущего элемента для плотных матриц и сравнить с прямым методом
5. Реализовать метод сопряженных градиентов, провести измерения на различных матрицах

2 Ход работы

2.1 Прямой метод решения СЛАУ на основе LU-разложения

- LU-разложение:

  $L_{11} = A_{11}$

  $\forall i = 2..n$

$$L_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik} \cdot U_{kj};\ j = \overline{1, j-1}$$

$$L_{ii} = A_{ii} - \sum_{k=1}^{j-1} L_{ik} \cdot U_{ki}$$

$$U_{ji} = \frac{1}{L_{jj}} \cdot \left[A_{ij} - \sum_{k=1}^{j-1} L_{jk} \cdot U_{ki} \right];\ j = \overline{1, i-1}$$

$$U_{ii} = 1$$

- Алгоритм:
  1. Разложить A на L, U
  2. Решить $Ly = b$ прямым ходом метода Гаусса
  3. Решить $Ux = y$ обратным ходом метода Гаусса
- Алгоритм метода Гаусса:
  1. Прямой ход

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \cdot a_{kj}^{(k-1)}$$

$$b_{j}^{(k)} = b_{j}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \cdot b_{k}^{(k-1)}$$

  где k - номер этапа: $\overline{1, n-1}$
  $i, j \in \overline{k+1, n}$

2. Обратный ход

$$x_k = \frac{1}{a_{kk}^{(k-1)}} \left[b_k^{(k-1)} - \sum_{j=k+1}^{n} a_{kj}^{(k-1)} \cdot x_j \right]$$

$k \in \overline{n, 1}$

- Общий вид генерируемых матриц:

$$a_{ii} = \begin{cases} -\sum_{i \neq j} a_{ij}, \ i > 1 \\ -\sum_{i \neq j} a_{ij} + 10^{-k}, \ i = 1 \end{cases}$$

a_{ij} выбирается случайно из $\{0, -1, -2, -3, -4\}$

$$n = 10$$

k	$\|x^* - x_k\|$	$\|x^* - x_k\| / \|x^*\|$	$cond$
0	2,884e-27	7,490e-30	1,721e-29
1	7,815e-25	2,030e-27	4,733e-27
2	1,044e-23	2,711e-26	6,333e-26
3	3,518e-20	9,138e-23	2,135e-22
4	2,322e-18	6,032e-21	1,409e-20
5	6,957e-16	1,807e-18	4,221e-18
6	1,374e-14	3,569e-17	8,338e-17
7	5,917e-12	1,537e-14	3,590e-14
8	7,730e-13	2,008e-15	4,690e-15
9	7,223e-09	1,876e-11	4,383e-11
10	1,279e-05	3,321e-08	7,762e-08
11	5,807e-05	1,508e-07	3,521e-07
12	1,951e-03	5,067e-06	1,179e-05
13	4,803e+00	1,248e-02	3,575e-02
14	9,972e+01	2,590e-01	1,945e-01
15	6,400e+02	1,662e+00	1,573e+00
16	6,400e+02	1,662e+00	1,573e+00
17	6,400e+02	1,662e+00	1,573e+00
18	6,400e+02	1,662e+00	1,573e+00
19	6,400e+02	1,662e+00	1,573e+00

$n = 100$

k	$\|x^* - x_k\|$	$\|x^* - x_k\|/\|x^*\|$	$cond$
0	1,675e-21	4,952e-27	4,389e-25
1	5,764e-19	1,703e-24	1,510e-22
2	7,783e-18	2,300e-23	2,039e-21
3	1,377e-14	4,071e-20	3,608e-18
4	9,523e-13	2,815e-18	2,494e-16
5	1,101e-10	3,253e-16	2,883e-14
6	2,589e-09	7,651e-15	6,780e-13
7	2,546e-06	7,525e-12	6,669e-10
8	7,976e-05	2,357e-10	2,089e-08
9	1,165e-02	3,443e-08	3,052e-06
10	1,084e+00	3,204e-06	2,829e-04
11	1,376e+03	4,068e-03	4,093e-01
12	2,063e+04	6,098e-02	3,070e+00
13	4,723e+06	1,396e+01	1,251e+04
14	6,241e+07	1,845e+02	4,088e+06
15	6,241e+07	1,845e+02	4,088e+06
16	6,241e+07	1,845e+02	4,088e+06
17	6,241e+07	1,845e+02	4,088e+06
18	6,241e+07	1,845e+02	4,088e+06
19	6,241e+07	1,845e+02	4,088e+06

$n = 500$

k	$\|x^* - x_k\|$	$\|x^* - x_k\|/\|x^*\|$	$cond$
0	3,742e-17	8,954e-25	9,486e-21
1	2,464e-19	5,896e-27	6,246e-23
2	4,012e-14	9,600e-22	1,017e-17
3	3,716e-11	8,891e-19	9,418e-15
4	2,283e-10	5,463e-18	5,787e-14
5	1,166e-07	2,790e-15	2,955e-11
6	2,417e-06	5,782e-14	6,125e-10
7	7,869e-04	1,883e-11	1,994e-07
8	1,751e-01	4,189e-09	4,438e-05
9	9,386e-01	2,246e-08	2,380e-04
10	1,467e+03	3,509e-05	3,678e-01
11	3,096e+05	7,407e-03	6,700e+01
12	6,733e+05	1,611e-02	1,348e+02
13	6,208e+07	1,485e+00	5,800e+03
14	5,205e+07	1,245e+00	3,991e+03
15	2,319e+07	5,549e-01	1,450e+03
16	2,319e+07	5,549e-01	1,450e+03
17	2,319e+07	5,549e-01	1,450e+03
18	2,319e+07	5,549e-01	1,450e+03
19	2,319e+07	5,549e-01	1,450e+03

$n = 1000$

k	$\|x^* - x_k\|$	$\|x^* - x_k\|/\|x^*\|$	$cond$
0	8,636e-16	2,587e-24	1,663e-19
1	3,992e-14	1,196e-22	7,684e-18
2	1,317e-12	3,945e-21	2,535e-16
3	3,560e-10	1,066e-18	6,852e-14
4	1,138e-08	3,408e-17	2,190e-12
5	6,419e-06	1,923e-14	1,236e-09
6	6,416e-04	1,922e-12	1,235e-07
7	3,206e-03	9,605e-12	6,172e-07
8	9,446e-02	2,829e-10	1,818e-05
9	1,436e+02	4,302e-07	2,761e-02
10	2,689e+02	8,056e-07	5,169e-02
11	1,283e+07	3,844e-02	1,714e+03
12	3,749e+05	1,123e-03	7,650e+01
13	3,241e+08	9,707e-01	1,566e+04
14	4,687e+08	1,404e+00	3,062e+04
15	4,687e+08	1,404e+00	3,062e+04
16	4,687e+08	1,404e+00	3,062e+04
17	4,687e+08	1,404e+00	3,062e+04
18	4,687e+08	1,404e+00	3,062e+04
19	4,687e+08	1,404e+00	3,062e+04

График зависимости логарифма относительной погрешности от k

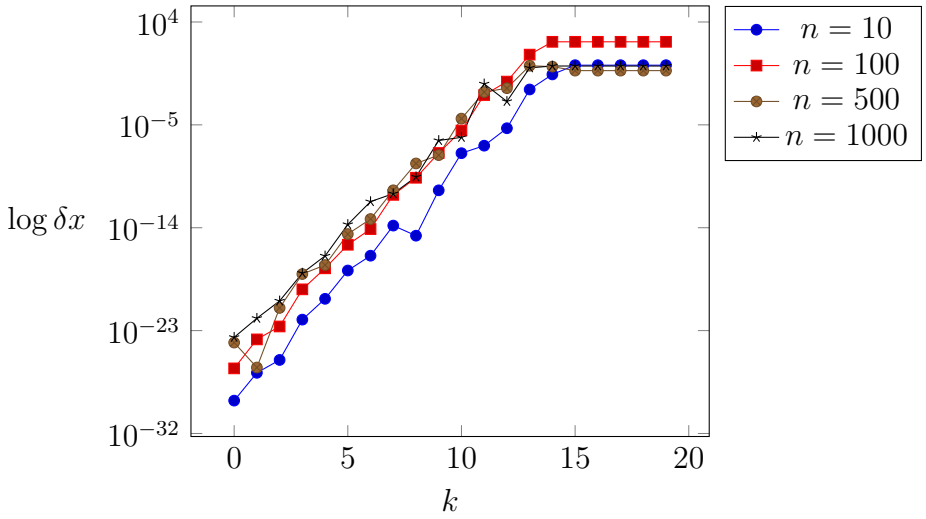
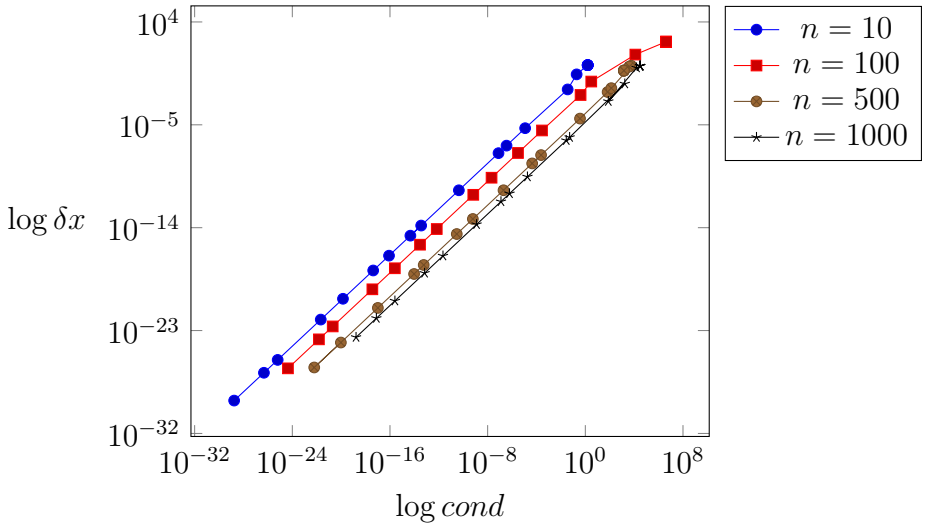


График зависимости логарифма относительной погрешности от *cond*



При увеличении k растет число обусловленности и экспоненциально уменьшается получаемая точность. При $k \geq 15$ погрешность перестает изменяться для всех рассмотренных размерностей и становится максимальной, т.к. точность ЭВМ не позволяет выразить различия между рассматриваемыми матрицами.

По результатам измерений при $k = 10$ вычитание $a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} \cdot a_{kj}^{(k-1)}$ близких друг к другу элементов вызывает скачкообразное накопление погрешности.

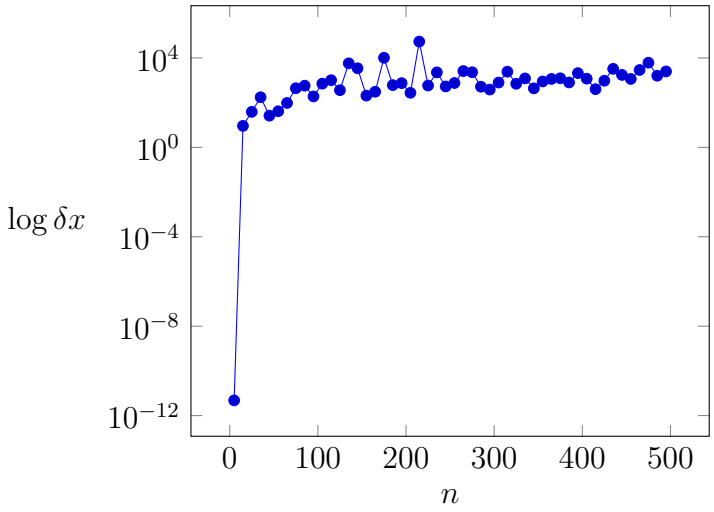
2.2 Исследования на матрицах Гильберта

Общий вид матрицы:

$$a_{ij} = \frac{1}{i + j - 1}, \ i, j \in \overline{1, k}$$

Размерность	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	$cond$
5	3,558e-11	4,798e-12	2,237e-12
15	3,219e+02	9,142e+00	4,749e-02
25	2,888e+03	3,886e+01	6,420e-03
35	2,131e+04	1,745e+02	3,685e-03
45	4,629e+03	2,612e+01	1,454e-04
55	9,868e+03	4,134e+01	8,837e-05
65	2,965e+04	9,688e+01	9,794e-05
75	1,677e+05	4,428e+02	2,433e-04
85	2,625e+05	5,750e+02	1,910e-04
95	1,024e+05	1,901e+02	4,150e-05
105	4,423e+05	7,070e+02	1,082e-04
115	7,260e+05	1,013e+03	1,141e-04
125	2,939e+05	3,621e+02	3,116e-05
135	5,234e+06	5,747e+03	3,884e-04
145	3,494e+06	3,448e+03	1,869e-04
155	2,316e+05	2,069e+02	9,144e-06
165	3,780e+05	3,075e+02	1,124e-05
175	1,365e+07	1,017e+04	3,112e-04
185	8,870e+05	6,081e+02	1,575e-05
195	1,178e+06	7,462e+02	1,652e-05
205	4,702e+05	2,765e+02	5,285e-06
215	9,867e+07	5,402e+04	8,993e-04
225	1,141e+06	5,838e+02	8,537e-06
235	4,717e+06	2,261e+03	2,927e-05
245	1,179e+06	5,309e+02	6,131e-06
255	1,790e+06	7,592e+02	7,872e-06
265	6,470e+06	2,590e+03	2,427e-05
275	6,071e+06	2,300e+03	1,959e-05
285	1,436e+06	5,155e+02	4,015e-06
295	1,131e+06	3,857e+02	2,760e-06
305	2,447e+06	7,936e+02	5,242e-06
315	7,868e+06	2,432e+03	1,489e-05
325	2,375e+06	7,006e+02	3,993e-06
335	4,303e+06	1,213e+03	6,457e-06
345	1,621e+06	4,371e+02	2,181e-06
355	3,434e+06	8,875e+02	4,163e-06
365	4,634e+06	1,149e+03	5,080e-06
375	5,192e+06	1,236e+03	5,167e-06
385	3,499e+06	8,008e+02	3,173e-06
395	9,511e+06	2,095e+03	7,882e-06
405	5,573e+06	1,182e+03	4,233e-06
415	1,960e+06	4,008e+02	1,368e-06
425	4,848e+06	9,567e+02	3,120e-06
435	1,710e+07	3,258e+03	1,017e-05
445	9,466e+06	1,744e+03	5,212e-06
455	6,467e+06	1,152e+03	3,304e-06
465	1,678e+07	2,894e+03	7,971e-06
475	3,673e+07	6,135e+03	1,625e-05
485	9,887e+06	1,601e+03	4,082e-06
495	1,587e+07	2,492e+03	6,123e-06

График зависимости логарифма относительной погрешности от размерности матрицы



3 Метод Гаусса с выбором ведущего элемента для плотных матриц

Необходимо найти $m \geq k$, где k — номер рассматриваемого шага, а $|a_{mk}| = \max_{i \geq k}\{|a_{ik}|\}$.

- Если $a_{mk} = 0 (\approx \varepsilon)$, однозначного решения нет, остановка алгоритма.
- Если $a_{mk} \neq 0$, меняем местами b_k и b_m; a_{kj} и a_{mj} при $j = k \ldots n$.

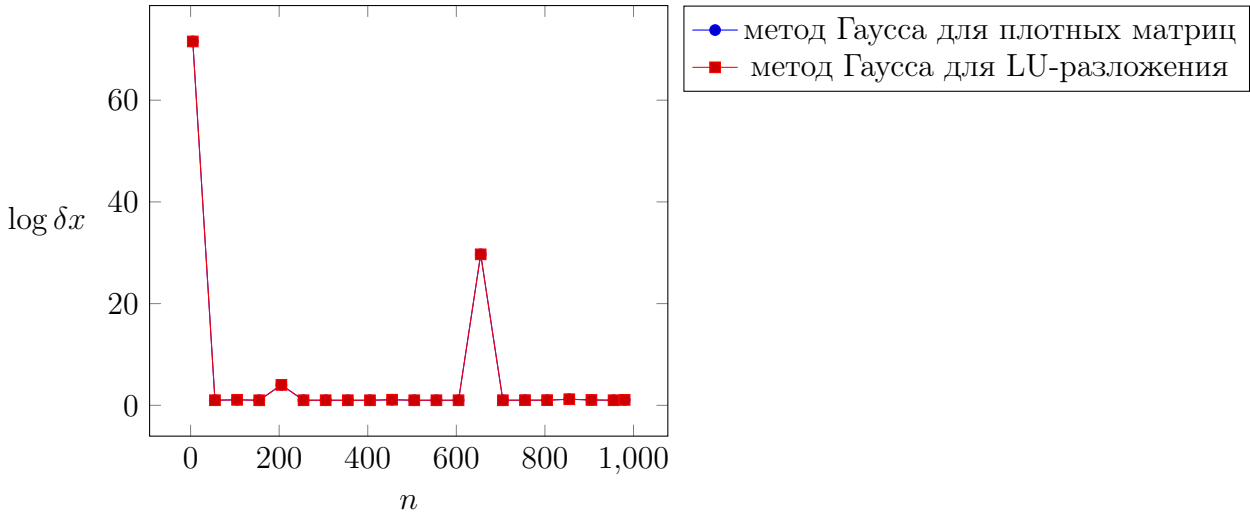
Метод Гаусса

Размерность	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	Количество действий
5	3,935e+03	7,155e+01	1,066e+02
55	5,732e+04	1,006e+00	1,030e+03
105	4,236e+05	1,082e+00	8,867e+03
155	1,250e+06	9,977e-01	8,285e+04
205	1,162e+07	4,016e+00	1,790e+09
255	5,577e+06	1,003e+00	1,377e+05
305	9,547e+06	1,005e+00	4,204e+05
355	1,496e+07	9,993e-01	6,832e+02
405	2,219e+07	9,986e-01	1,043e+03
455	3,457e+07	1,097e+00	2,474e+04
505	4,302e+07	9,992e-01	9,644e+01
555	5,707e+07	9,988e-01	3,606e+02
605	7,393e+07	9,991e-01	8,748e+02
655	2,787e+09	2,969e+01	1,181e+10
705	1,171e+08	1,000e+00	1,896e+03
755	1,457e+08	1,014e+00	3,568e+05
805	1,768e+08	1,015e+00	5,102e+03
855	2,501e+08	1,198e+00	4,361e+04
905	2,607e+08	1,053e+00	5,017e+05
955	2,917e+08	1,003e+00	3,707e+03
980	3,356e+08	1,068e+00	8,922e+03

Прямой метод LU-разложения

Размерность	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	Количество действий
5	3,935e+03	7,155e+01	6,293e+06
55	5,732e+04	1,006e+00	1,930e+02
105	4,236e+05	1,082e+00	7,607e+05
155	1,250e+06	9,977e-01	3,480e+05
205	1,162e+07	4,016e+00	9,916e+10
255	5,577e+06	1,003e+00	1,410e+05
305	9,547e+06	1,005e+00	1,177e+05
355	1,496e+07	9,993e-01	6,499e+03
405	2,219e+07	9,986e-01	9,724e+02
455	3,457e+07	1,097e+00	6,717e+04
505	4,302e+07	9,992e-01	5,042e+02
555	5,707e+07	9,988e-01	3,604e+03
605	7,393e+07	9,991e-01	6,061e+03
655	2,787e+09	2,969e+01	1,286e+10
705	1,171e+08	1,000e+00	1,864e+04
755	1,457e+08	1,014e+00	2,374e+05
805	1,768e+08	1,015e+00	1,122e+04
855	2,501e+08	1,198e+00	3,384e+05
905	2,607e+08	1,053e+00	5,516e+05
955	2,917e+08	1,003e+00	4,133e+04
980	3,356e+08	1,068e+00	2,035e+05

График зависимости логарифма относительной погрешности от размерности матрицы



Два графика наложились друг на друга из-за близких значений погрешности.

4 Метод сопряженных градиентов

Точность решения СЛАУ 10^{-7}

4.1 Матрица с диагональным преобладанием

Общий вид матрицы:

$$a_{ii} = \begin{cases} -\sum_{i \neq j} a_{ij}, \ i > 1 \\ -\sum_{i \neq j} a_{ij} + 1, \ i = 1 \end{cases}$$

Размерность	Количество итераций	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	$cond$
10	10	1,164e-29	3,022e-32	1,125e-63
16	15	6,032e-14	4,032e-17	1,150e-32
26	19	1,867e-13	3,011e-17	4,189e-32
42	24	1,472e-13	5,753e-18	4,747e-33
69	27	7,172e-13	6,409e-18	1,927e-32
113	34	1,180e-12	2,422e-18	9,496e-33
186	46	2,689e-13	1,244e-19	9,643e-35
306	63	3,042e-12	3,170e-19	1,475e-33
504	95	1,600e-12	3,738e-20	5,754e-35
830	145	9,291e-12	4,866e-20	2,784e-34
1368	227	4,026e-12	4,713e-21	7,613e-36
2255	360	3,471e-11	9,076e-21	7,525e-35
3717	581	4,473e-11	2,612e-21	8,050e-36
6128	958	6,300e-11	8,211e-22	2,253e-36
10103	1567	4,152e-10	1,208e-21	1,005e-35
16657	2579	1,088e-09	7,062e-22	4,966e-36
27462	4240	3,139e-09	4,547e-22	2,765e-36
45277	7011	1,214e-08	3,922e-22	2,731e-36
74649	11524	2,537e-08	1,829e-22	1,487e-36
100000	17573	2,832e-08	8,495e-23	5,691e-37

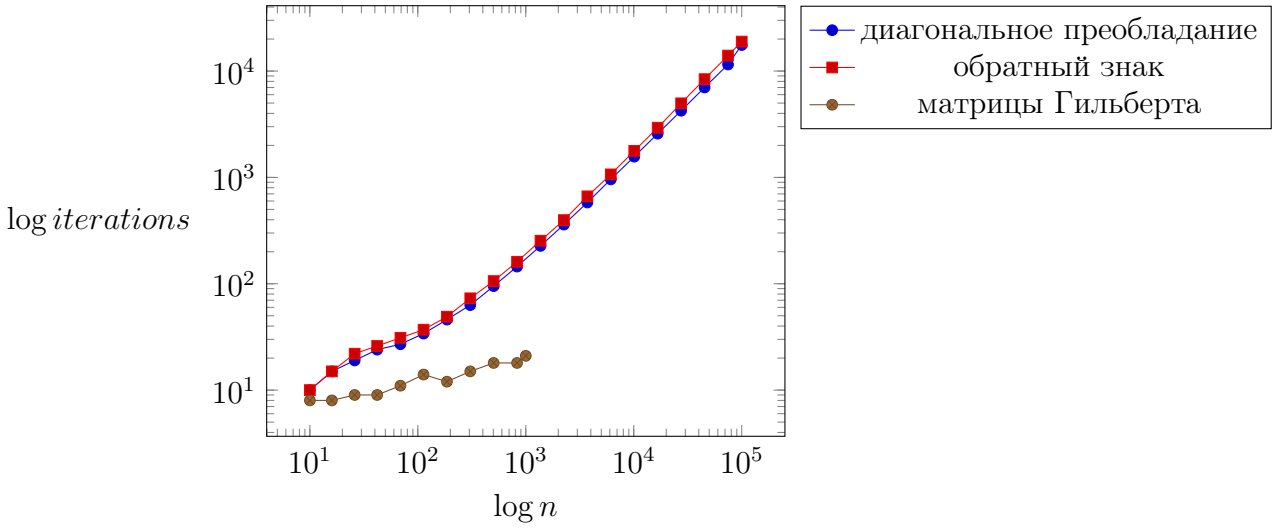
4.2 Матрица с обратным знаком внедиагональных элементов

Размерность	Количество итераций	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	$cond$
10	10	1,545e-25	4,012e-28	1,086e-55
16	15	1,936e-12	1,294e-15	1,060e-29
26	22	1,861e-14	3,002e-18	5,249e-34
42	26	3,004e-13	1,174e-17	1,460e-32
69	31	1,413e-12	1,263e-17	5,968e-32
113	37	1,409e-12	2,891e-18	8,005e-33
186	49	5,318e-12	2,459e-18	1,861e-32
306	73	3,105e-12	3,235e-19	6,979e-34
504	106	1,799e-11	4,204e-19	3,389e-33
830	161	1,587e-09	8,314e-18	5,317e-32
1368	254	4,199e-09	4,915e-18	9,940e-33
2255	397	3,023e-06	7,903e-16	4,720e-30
3717	664	2,295e-08	1,340e-18	4,571e-33
6128	1068	1,060e-09	1,382e-20	1,208e-34
10103	1776	6,087e-06	1,771e-17	1,210e-31
16657	2931	7,681e-06	4,986e-18	4,403e-32
27462	4974	3,711e-05	5,376e-18	5,241e-32
45277	8395	8,955e-04	2,894e-17	2,882e-31
74649	13993	2,494e-01	1,798e-15	1,719e-29
100000	18932	2,888e-02	8,663e-17	7,693e-31

4.3 Матрица Гильберта

Размерность	Количество итераций	$\|x^* - x\|$	$\|x^* - x\|/\|x^*\|$	$cond$
10	8	6,408e-02	3,266e-03	7,980e+06
16	8	3,907e-01	1,010e-02	5,256e+05
26	9	5,697e-01	7,235e-03	8,607e+05
42	9	2,318e+00	1,449e-02	1,821e+05
69	11	3,546e+00	1,060e-02	3,416e+05
113	14	5,644e+00	8,084e-03	5,808e+05
186	12	1,955e+01	1,330e-02	1,581e+05
306	15	3,205e+01	1,035e-02	2,741e+05
504	18	5,429e+01	8,299e-03	4,242e+05
830	18	1,678e+02	1,215e-02	1,597e+05
1000	21	1,440e+02	7,884e-03	4,207e+05

График зависимости количества итераций метода от размерности



Матрица Гильберта

Искомая точность не достигается, т.к. МСГ использует невязку как критерий останова, и искомая невязка достигается. Вследствие этого выполняется малое число итераций.

Прочие матрицы

Точность полученного решения задается параметром ε, поэтому во всех измерениях, независимо от размерности и типа, наблюдается почти одинаковое значение погрешности.
Знак внедиагональных элементов для сгенерированных матриц не влияет на количество итераций.
Количество итераций линейно зависит от размерности матрицы.

5 Выводы

1. Реализованы следующие алгоритмы решения СЛАУ:
   - Метод Гаусса для LU-разложения для профильных матриц
   - Метод Гаусса с выбором ведущего элемента для плотных матриц
   - Метод сопряженных градиентов для симметричных матриц
2. Для метода Гаусса с LU-разложением была установлена зависимость точности алгоритма от размерности матрицы и линейная зависимость от числа обусловленности.
3. Для плохо обусловленных матриц Гильберта методы Гаусса дают ответ, отличающийся от правильного не больше, чем на 10^7
4. Метод Гаусса с LU-разложением и метод Гаусса с выбором ведущего элемента показали одинаковую точность, но LU разложение потребовало больше действий.
5. Метод сопряженных градиентов считает ответ с небольшой погрешностью, но работает долго на больших размерностях. Для матриц Гильберта метод сопряженных градиентов не достигает искомой точности.

6 Исходный код

Вектор

```
1  package matrix;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Vector {
7
8      public Vector(List<Double> list) {
9          values = list;
10         dim = list.size();
11     }
12
13     public double get(int index) {
14         return values.get(index - 1);
15     }
16
17     public void set(int index, double value) {
18         values.set(index - 1, value);
19     }
20
21     public double mulVector(Vector other) {
22         assert (other.dim == dim);
23         double res = 0;
24         for (int i = 1; i <= dim; i++) {
25             res += other.get(i) * get(i);
26         }
27         return res;
28     }
29
30     public Vector addVector(Vector other) {
31         assert (other.dim == dim);
```

```
        List<Double> res = new ArrayList<>();
        for (int i = 1; i <= dim; i++) {
            res.add(get(i) + other.get(i));
        }
        return new Vector(res);
    }

    public Vector subVector(Vector other) {
        return addVector(other.mulNumber(-1));
    }

    public Vector mulNumber(double value) {
        List<Double> res = new ArrayList<>();
        for (double d : values) {
            res.add(d * value);
        }
        return new Vector(res);
    }

    public double norm() {
        double res = 0;
        for (double value : values) {
            res += value * value;
        }
        return Math.sqrt(res);
    }

    @Override
    public String toString() {
        StringBuilder res = new StringBuilder();
        for (double value : values) {
            res.append(value).append(" ");
        }
        return res.toString();
    }

    public int size() {
        return dim;
    }

    public Vector copy() {
        return new Vector(getValues());
    }

    public List<Double> getValues() {
        return new ArrayList<>(values);
    }

    private final List<Double> values;
    private final int dim;
}
```

Общий класс матриц

```java
package matrix;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import static matrix.Utils.parseDoubleList;

/**
 * Абстрактный класс матрицы
 */
public abstract class Matrix {

    Matrix(int dim) {
        this.dim = dim;
        index = IntStream.range(1, dim + 1)
                .boxed()
                .collect(Collectors.toList());
    }

    /**
     * @return Матрица в виде двумерного списка
     */
    public List<List<Double>> to2DList() {
        List<List<Double>> res = new ArrayList<>();
        for (int i = 1; i <= dim; i++) {
            res.add(new ArrayList<>());
            for (int j = 1; j <= dim; j++) {
                res.get(i - 1).add(get(i, j));
            }
        }
        return res;
    }

    /**
     * @return A * b
     */
    public List<Double> mulVector(List<? extends Number> b) {
        List<Double> res = new java.util.ArrayList<>(Collections.nCopies(dim, 0.));
        for (int i = 0; i < dim; i++) {
            for (int j = 0; j < dim; j++) {
                res.set(i, res.get(i) + get(i + 1, j + 1) * b.get(j).doubleValue());
            }
        }
        return res;
    }

    public double get(int row, int column) {
        return getImpl(index.get(row - 1), column);
    }

```

```
57      public void set(int row, int column, Number value) {
58          setImpl(index.get(row - 1), column, value);
59      }
60
61      /**
62       * Прямой шаг метода Гаусса
63       *
64       * @param b
65       * @param modified Использовать ли выбор ведущего элемента
66       */
67      public List<Double> directMethod(List<Double> b, boolean modified) {
68          List<Double> b1 = new ArrayList<>(b);
69          for (int k = 1; k <= dim - 1; k++) {
70              if (modified) {
71                  swap(k, b1);
72              }
73              for (int i = k + 1; i <= dim; i++) {
74                  double tIK = get(i, k) / get(k, k);
75                  b1.set(i - 1, b1.get(i - 1) - tIK * b1.get(k - 1));
76                  for (int j = k; j <= dim; j++) {
77                      set(i, j, get(i, j) - tIK * get(k, j));
78                  }
79              }
80          }
81          return b1;
82      }
83
84      /**
85       * Обратный шаг метода Гаусса
86       *
87       * @param b
88       */
89      public List<Double> reverseMethod(List<Double> b) {
90          List<Double> x = new ArrayList<>(Collections.nCopies(dim, 0.));
91          x.set(dim - 1, b.get(dim - 1) / get(dim, dim));
92          for (int k = dim - 1; k >= 1; k--) {
93              double temp = b.get(k - 1);
94              for (int j = k + 1; j <= dim; j++) {
95                  temp -= get(k, j) * x.get(j - 1);
96              }
97              x.set(k - 1, temp / get(k, k));
98          }
99          return x;
100     }
101
102     /**
103      * Метод Гаусса
104      *
105      * @param b
106      * @param modified Использовать ли выбор ведущего элемента
107      * @return x такое, что Ax = b
108      */
109     public List<Double> gauss(List<Double> b, boolean modified) {
110         return reverseMethod(directMethod(b, modified));
111     }
112
113     /**
114      * Метод Гаусса
115      *
116      * @param b
```

```
117         * @return x такое, что Ax = b
118         */
119        public List<Double> gauss(List<Double> b) {
120            return gauss(b, false);
121        }
122
123        /**
124         * Вывод матрицы в консоль
125         */
126        public void print() {
127            print(Collections.emptyList());
128        }
129
130        /**
131         * Вывод матрицы в консоль
132         */
133        public void print(List<?> b) {
134            for (int i = 1; i <= dim; i++) {
135                for (int j = 1; j <= dim; j++) {
136                    System.out.print(get(i, j) + " ");
137                }
138                if (b.isEmpty()) {
139                    System.out.println();
140                } else {
141                    System.out.println(b.get(i - 1));
142                }
143            }
144        }
145
146        public abstract double getImpl(int row, int column);
147
148        public abstract void setImpl(int row, int column, Number value);
149
150        /**
151         * Список переставленных строк
152         */
153        protected final List<Integer> index;
154
155        /**
156         * Размерность матрицы
157         */
158        protected final int dim;
159
160        protected void swap(int row, List<Double> b) {
161            int minRow = row;
162            for (int i = row; i <= dim; i++) {
163                if (Math.abs(get(i, row)) > Math.abs(get(minRow, row))) {
164                    minRow = i;
165                }
166            }
167            int temp = index.get(row - 1);
168            index.set(row - 1, index.get(minRow - 1));
169            index.set(minRow - 1, temp);
170            double temp2 = b.get(row - 1);
171            b.set(row - 1, b.get(minRow - 1));
172            b.set(minRow - 1, temp2);
173        }
174    }
```

Профильная матрица

```java
package matrix;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javafx.util.Pair;

import static matrix.Utils.parseDoubleList;
import static matrix.Utils.parseIntegerList;

/**
 * Профильная матрица
 */
public class ProfileMatrix extends Matrix {

    public ProfileMatrix(BufferedReader reader) throws IOException {
        super(Integer.parseInt(reader.readLine()));
        di = parseDoubleList(reader);
        al = parseDoubleList(reader);
        au = parseDoubleList(reader);
        ia = parseIntegerList(reader);
    }

    @Override
    public List<Double> gauss(List<Double> b, boolean modified) {
        if (modified) {
            throw new
            ↪  UnsupportedOperationException("modified gauss is used only on dense matrices");
        }
        return reverseMethod(directMethod(b, false));
    }

    @Override
    public List<Double> mulVector(List<? extends Number> b) {
        List<Double> res = new ArrayList<>(Collections.nCopies(dim, 0.));
        if ((dim != b.size()))
            throw new IllegalStateException();
        for (int i = 0; i < dim; i++) {
            int count = ia.get(i + 1) - ia.get(i);
            res.set(i, res.get(i) + di.get(i) * b.get(i).doubleValue());
            for (int j = i - count, index = ia.get(i) - 1; j < i; j++, index++) {
                res.set(i, res.get(i) + al.get(index) * b.get(j).doubleValue());
                res.set(j, res.get(j) + au.get(index) * b.get(i).doubleValue());
            }
        }
        return res;
    }

    @Override
    public double getImpl(int row, int column) {
        return getImpl(indexPair(row, column));
    }

    @Override
```

```
56      public void setImpl(int row, int column, Number value) {
57          var indexPair = indexPair(row, column);
58          if (indexPair == null && value.doubleValue() == 0) {
59              return;
60          }
61          assert (indexPair != null);
62          int lDUIndex = indexPair.getKey();
63          switch (indexPair.getValue()) {
64              case 0 -> al.set(lDUIndex, value.doubleValue());
65              case 1 -> di.set(lDUIndex, value.doubleValue());
66              case 2 -> au.set(lDUIndex, value.doubleValue());
67              default -> throw new IllegalStateException("Invalid indexPair");
68          }
69      }
70
71      /**
72       * @return Пара из (a, b), где b - в котором из (al, di, au) брать искомый
73       *         элемент, а a - индекс в соответствующем массиве
74       */
75      private Pair<Integer, Integer> indexPair(int row, int column) {
76          row--;
77          column--;
78          if (row == column) {
79              return new Pair<>(row, 1);
80          }
81          List<Double> aul = al;
82          if (row < column) {
83              aul = au;
84              int temp = row;
85              row = column;
86              column = temp;
87          }
88          int add = column - (row - (ia.get(row + 1) - ia.get(row)));
89          if (add < 0) {
90              return null;
91          }
92          return new Pair<>(ia.get(row) - 1 + add, (aul == al ? 0 : 2));
93      }
94
95      private double getImpl(Pair<Integer, Integer> indexPair) {
96          if (indexPair == null) {
97              return 0;
98          }
99          int lDUIndex = indexPair.getKey();
100         return switch (indexPair.getValue()) {
101             case 0 -> al.get(lDUIndex);
102             case 1 -> di.get(lDUIndex);
103             case 2 -> au.get(lDUIndex);
104             default -> throw new IllegalStateException("Invalid indexPair");
105         };
106     }
107
108     private final List<Double> di, al, au;
109     private final List<Integer> ia;
110 }
```

Плотная матрица

```java
package matrix;

import java.util.ArrayList;
import java.util.List;

/**
 * Плотная матрица
 */
public class DenseMatrix extends Matrix {
    DenseMatrix(List<List<Double>> matrix) {
        super(matrix.size());
        this.matrix = matrix;
    }

    /**
     * @return Эта матрица, умноженная на транспозицию себя
     */
    public DenseMatrix mulMatrixWithTransposeSelf() {
        List<List<Double>> res = new ArrayList<>();
        for (int i = 0; i < dim; i++) {
            res.add(new ArrayList<>());
            for (int j = 0; j < dim; j++) {
                res.get(i).add(0.);
            }
        }

        for (int i = 1; i <= dim; i++) {
            for (int j = 1; j <= dim; j++) {
                for (int r = 1; r <= dim; r++) {
                    res.get(i - 1).set(j - 1, res.get(i - 1).get(j - 1) + get(r, i) *
                    ↪  get(r, j));
                }
            }
        }
        return new DenseMatrix(res);
    }

    @Override
    public void setImpl(int row, int column, Number value) {
        matrix.get(row - 1).set(column - 1, value.doubleValue());
    }

    @Override
    public double getImpl(int row, int column) {
        return matrix.get(row - 1).get(column - 1);
    }

    private final List<List<Double>> matrix;
}
```

Разреженная матрица

```java
package matrix;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import javafx.util.Pair;

import static matrix.Utils.parseDoubleList;
import static matrix.Utils.parseIntegerList;

public class SparseMatrix extends Matrix {

    public SparseMatrix(BufferedReader reader) throws IOException {
        super(Integer.parseInt(reader.readLine()));
        di = parseDoubleList(reader);
        al = parseDoubleList(reader);
        au = parseDoubleList(reader);
        ia = parseIntegerList(reader);
        ja = parseIntegerList(reader);
    }

    public SparseMatrix(List<List<Double>> source) {
        super(source.size());
        di = new ArrayList<>();
        al = new ArrayList<>();
        au = new ArrayList<>();
        ia = new ArrayList<>();
        ja = new ArrayList<>();
        ia.add(1);
        for (int i = 1; i <= dim; i++) {
            ia.add(ia.get(ia.size() - 1));
            int temp = 0;
            for (int j = 1; j < i; j++) {
                double d = source.get(i - 1).get(j - 1);
                if (d == 0) {
                    continue;
                }
                temp++;
                al.add(d);
                au.add(source.get(j - 1).get(i - 1));
                ja.add(j);
            }
            ia.set(ia.size() - 1, ia.get(ia.size() - 1) + temp);
            di.add(source.get(i - 1).get(i - 1));
        }
    }

    public Vector msg(Vector f, double eps) {
        Vector
                x = new Vector(Collections.nCopies(dim, 1.)),
                r = f.subVector(mulVector(x)),
                z = r.copy();
        for (int k = 1; k <= MAX_ITERATIONS && r.norm() / f.norm() > eps; k++) {
```

```
57              Vector rOld = r.copy();
58              double a = r.mulVector(r) / mulVector(z).mulVector(z);
59              x = x.addVector(z.mulNumber(a));
60              r = r.subVector(mulVector(z).mulNumber(a));
61              double b = r.mulVector(r) / rOld.mulVector(rOld);
62              z = r.addVector(z.mulNumber(b));
63          }
64          return x;
65      }
66
67      public Vector mulVector(Vector b) {
68          List<Double> res = new ArrayList<>(Collections.nCopies(dim, 0.));
69          if ((dim != b.size())) throw new IllegalStateException();
70          for (int i = 0; i < dim; i++) {
71              int count = ia.get(i + 1) - ia.get(i);
72              res.set(i, res.get(i) + di.get(i) * b.get(i + 1));
73              for (int index = ia.get(i) - 1, indexLast = index + count; index <
                ↪  indexLast; index++) {
74                  int j = ja.get(index) - 1;
75                  res.set(i, res.get(i) + al.get(index) * b.get(j + 1));
76                  res.set(j, res.get(j) + au.get(index) * b.get(i + 1));
77              }
78          }
79          return new Vector(res);
80      }
81
82      @Override
83      public double getImpl(int row, int column) {
84          return getImpl(indexPair(row, column));
85      }
86
87      @Override
88      public void setImpl(int row, int column, Number value) {
89          var indexPair = indexPair(row, column);
90          if (indexPair == null && value.doubleValue() == 0) {
91              return;
92          }
93          assert (indexPair != null);
94          int lDUIndex = indexPair.getKey();
95          switch (indexPair.getValue()) {
96              case 0 -> al.set(lDUIndex, value.doubleValue());
97              case 1 -> di.set(lDUIndex, value.doubleValue());
98              case 2 -> au.set(lDUIndex, value.doubleValue());
99              default -> throw new IllegalStateException("Invalid indexPair");
100         }
101     }
102
103     private double getImpl(Pair<Integer, Integer> indexPair) {
104         if (indexPair == null) {
105             return 0;
106         }
107         int lDUIndex = indexPair.getKey();
108         return switch (indexPair.getValue()) {
109             case 0 -> al.get(lDUIndex);
110             case 1 -> di.get(lDUIndex);
111             case 2 -> au.get(lDUIndex);
112             default -> throw new IllegalStateException("Invalid indexPair");
113         };
114     }
115
```

```
116         /**
117          * @return Пара из (a, b), где b - в котором из (al, di, au)
118          * брать искомый элемент, а a - индекс в соответствующем массиве
119          */
120         private Pair<Integer, Integer> indexPair(int row, int column) {
121             row--;
122             column--;
123             if (row == column) {
124                 return new Pair<>(row, 1);
125             }
126             List<Double> aul = al;
127             if (row < column) {
128                 aul = au;
129                 int temp = row;
130                 row = column;
131                 column = temp;
132             }
133             int count = ia.get(row + 1) - ia.get(row);
134             for (int i = ia.get(row) - 1, lastI = i + count; i < lastI; i++) {
135                 if (column + 1 == ja.get(i)) {
136                     return new Pair<>(i, (aul == al ? 0 : 2));
137                 }
138             }
139             return null;
140         }
141 
142         private static final int MAX_ITERATIONS = 50000;
143         private final List<Double> di, al, au;
144         private final List<Integer> ia, ja;
145     }
```

LU-разложение матрицы

```java
package matrix;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class LUDecomposition extends Matrix {

    public LUDecomposition(ProfileMatrix matrix) {
        super(matrix.dim);
        this.matrix = matrix;
        decompose();
    }

    @Override
    public List<Double> directMethod(List<Double> b_, boolean modified) {
        List<Double> b = new ArrayList<>(b_);
        for (int k = 1; k <= dim - 1; k++) {
            if (modified) {
                swap(k, b);
            }
            for (int i = k + 1; i <= dim; i++) {
                double tIK = getL(i, k) / getL(k, k);
                b.set(i - 1, b.get(i - 1) - tIK * b.get(k - 1));
                for (int j = k; j <= i; j++) {
                    setL(i, j, getL(i, j) - tIK * getL(k, j));
                }
            }
        }
        for (int k = 1; k <= dim; k++) {
            b.set(k - 1, b.get(k - 1) / getL(k, k));
        }
        return b;
    }

    @Override
    public List<Double> reverseMethod(List<Double> b) {
        List<Double> x = new ArrayList<>(Collections.nCopies(dim, 0.));
        x.set(dim - 1, b.get(dim - 1) / getU(dim, dim));
        for (int k = dim - 1; k >= 1; k--) {
            double temp = b.get(k - 1);
            for (int j = k + 1; j <= dim; j++) {
                temp -= getU(k, j) * x.get(j - 1);
            }
            x.set(k - 1, temp / getU(k, k));
        }
        return x;
    }

    @Override
    public double getImpl(int row, int column) {
        return matrix.get(row, column);
    }

    @Override
    public void setImpl(int row, int column, Number value) {
```

```
57          matrix.set(row, column, value);
58      }
59
60      public double getL(int row, int column) {
61          if (row < column) {
62              return 0;
63          }
64          return matrix.get(row, column);
65      }
66
67      public double getU(int row, int column) {
68          if (row > column) {
69              return 0;
70          }
71          if (row == column) {
72              return 1;
73          }
74          return matrix.get(row, column);
75      }
76
77      public void setL(int row, int column, Number value) {
78          expectL(row, column);
79          matrix.set(row, column, value);
80      }
81
82      public void setU(int row, int column, Number value) {
83          expectR(row, column);
84          matrix.set(row, column, value);
85      }
86
87      private void expectL(int row, int column) {
88          if (row < column) {
89              throw new IllegalArgumentException("row must be >= column");
90          }
91      }
92
93      private void expectR(int row, int column) {
94          if (row > column) {
95              throw new IllegalArgumentException("row must be <= column");
96          }
97      }
98
99      /**
100      * Процедура LU-разложения
101      */
102     private void decompose() {
103         double temp;
104         for (int i = 2; i <= dim; i++) {
105             // Lij
106             for (int j = 1; j <= i - 1; j++) {
107                 temp = matrix.get(i, j);
108                 for (int k = 1; k <= j - 1; k++) {
109                     temp -= getL(i, k) * getU(k, j);
110                 }
111                 setL(i, j, temp);
112             }
113             // Uji
114             for (int j = 1; j <= i - 1; j++) {
115                 temp = matrix.get(j, i);
116                 for (int k = 1; k <= j - 1; k++) {
```

```
                    temp -= getL(j, k) * getU(k, i);
                }
                setU(j, i, temp / getL(j, j));
            }
            // Lii
            temp = matrix.get(i, i);
            for (int k = 1; k <= i - 1; k++) {
                temp -= getL(i, k) * getU(k, i);
            }
            setL(i, i, temp);
        }
    }

    public final Matrix matrix;
}
```

Генератор матриц

```java
package matrix;

import com.beust.jcommander.JCommander;
import com.beust.jcommander.Parameter;
import org.apache.commons.io.FileUtils;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import java.util.function.BiFunction;
import java.util.function.Function;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import static java.lang.Math.pow;
import static matrix.Utils.*;

/**
 * Абстрактный класс генератора матрицы
 */
public abstract class Generator {

    Generator(Args args) {
        this.args = args;
    }

    /**
     * Фабрика матриц
     */
    abstract Matrix readMatrix(BufferedReader reader) throws IOException;

    abstract void generate(BiFunction<Integer, Integer, Double> aValueGenerator,
                           Function<Integer, Double> bValueGenerator) throws
                           ↪  IOException;

    /**
     * Очищает и создает папку
     */
    public void prepareDir() throws IOException {
        final var testDirectory = Path.of(args.root, args.testName);
        FileUtils.deleteDirectory(new File(testDirectory.toString()));
        Files.createDirectories(testDirectory);
    }

    /**
     * Процедура генерации матрицы
     */
    protected void generate() throws IOException {
        switch (args.type) {
            case "int" -> generateInt();
```

```
            case "random" -> generateRandom();
            case "hilbert" -> generateHilbert();
            case "seq" -> generateSequence();
            default -> throw new IllegalStateException("Unexpected type: " + args.type);
        }
    }

    protected void generateRandom() throws IOException {
        generate((i, j) -> ThreadLocalRandom.current().nextDouble(args.min, args.max),
                i -> ThreadLocalRandom.current().nextDouble(args.min, args.max));
    }

    protected void generateInt() throws IOException {
        generate((i, j) -> (double) ThreadLocalRandom.current().nextInt((int) args.min,
        ↪  (int) args.max),
                i -> (double) ThreadLocalRandom.current().nextInt((int) args.min, (int)
                ↪  args.max));
    }

    protected void generateHilbert() throws IOException {
        args.zeroProbability = 0;
        args.belowDiagsCount = args.dim;
        generate((i, j) -> 1. / (i + j - 1), i ->
        ↪  ThreadLocalRandom.current().nextDouble(args.min, args.max));

        var A = readMatrix(Files.newBufferedReader(Path.of(args.root, args.testName,
        ↪  "A.txt")));
        Files.delete(Path.of(args.root, args.testName, "b.txt"));
        writeList(A.mulVector(IntStream.rangeClosed(1,
        ↪  args.dim).boxed().collect(Collectors.toList())), "b");
    }

    protected double valueOrZero(double value, double zeroProbability) {
        if (ThreadLocalRandom.current().nextDouble(0, 1) < zeroProbability) {
            return 0;
        }
        return value;
    }

    protected double valueNonZero(BiFunction<Integer, Integer, Double> doubleGenerator,
    ↪  int i, int j) {
        double res = doubleGenerator.apply(i, j);
        while (res == 0) {
            res = doubleGenerator.apply(i, j);
        }
        return res;
    }

    protected void generateSequence() throws IOException {
        main("-t", "int", "--min", "-4", "--max", "0", "-n", "tmp", "-d",
        ↪  String.valueOf(args.dim), "-c",
                String.valueOf(args.belowDiagsCount), "--kind", args.kind);
        final String originalTestName = args.testName;
        for (int k = 0; k <= args.maxk; k++) {
            var APath = Path.of(args.root, "tmp", "A.txt");
            var A = readMatrix(Files.newBufferedReader(APath));

            var di = new ArrayList<Double>();
            for (int i = 1; i <= args.dim; i++) {
                double sum = 0;
```

```
109                 for (int j = 1; j <= args.dim; j++) {
110                     if (i == j) {
111                         continue;
112                     }
113                     sum += A.get(i, j);
114                 }
115                 if (i == 1) {
116                     sum -= pow(10, -k);
117                 }
118                 di.add(-sum);
119             }
120 
121             try (var reader = Files.newBufferedReader(APath)) {
122                 args.testName = originalTestName + "_" + k;
123                 prepareDir();
124                 writeList(parseIntegerList(reader), "A");
125                 writeList(di, "A");
126                 parseDoubleList(reader); // old di ignored
127                 writeList(parseDoubleList(reader), "A");
128                 writeList(parseDoubleList(reader), "A");
129                 while (!eof(reader)) {
130                     writeList(parseIntegerList(reader), "A");
131                 }
132             }
133 
134             var newA = readMatrix(Files.newBufferedReader(Path.of(args.root,
                ↪  args.testName, "A.txt")));
135             writeList(newA.mulVector(IntStream.rangeClosed(1,
                ↪  args.dim).boxed().collect(Collectors.toList())), "b");
136         }
137     }
138 
139     /**
140      * Записывает список в файл с именем {@code name} в папку {@code root}
141      */
142     protected <T> void writeList(List<T> list, String name) throws IOException {
143         Files.writeString(Path.of(args.root, args.testName, name + ".txt"),
144                 list.stream().map(Object::toString).collect(Collectors.joining(" ", "",
                    ↪  "\n")),
145                 StandardOpenOption.APPEND, StandardOpenOption.WRITE,
                    ↪  StandardOpenOption.CREATE);
146     }
147 
148     protected static class Args {
149         @Parameter(names = {"-r", "--root"}, description = "Корень всех тестов")
150         protected String root = "data";
151 
152         @Parameter(names = {"-n", "--test_name"}, description =
            ↪  "Название теста и соотвествующей папки")
153         protected String testName;
154 
155         @Parameter(names = {"-h", "--help"}, help = true, description =
            ↪  "Вывести помощь")
156         protected boolean help;
157 
158         @Parameter(names = {"-d", "--dimensionality"}, description = "Размерность")
159         protected int dim = 6;
160 
161         @Parameter(names = {"-c", "--diag_count"}, description =
            ↪  "Количество ненулевых диагоналей")
```

```
162         protected int belowDiagsCount = 2;
163
164         @Parameter(names = {"-z", "--zero_probability"}, description =
            ↪  "Вероятность элемента матрицы быть нулём")
165         protected double zeroProbability = 0.2;
166
167         @Parameter(names = "--min", description = "Минимальное значение элемента")
168         protected double min = -4;
169
170         @Parameter(names = "--max", description = "Максимальное значение элемента")
171         protected double max = 0;
172
173         @Parameter(names = {"-t", "--type"}, description = "Тип генератора")
174         protected String type = "int";
175
176         @Parameter(names = "--maxk", description = "Максимальное k")
177         protected int maxk = 10;
178
179         @Parameter(names = {"-k", "--kind"}, description =
            ↪  "Вид генерируемой матрицы (sparse, profile)")
180         protected String kind;
181     }
182
183     protected final Args args;
184
185     public static void main(String... args) throws IOException {
186         var argv = new Args();
187         var jc = new JCommander(argv);
188         jc.parse(args);
189
190         if (argv.help) {
191             jc.usage();
192             return;
193         }
194
195         switch (argv.kind) {
196             case "sparse" -> new SparseMatrixGenerator(argv).generate();
197             case "profile" -> new ProfileMatrixGenerator(argv).generate();
198             default -> throw new IllegalArgumentException(argv.kind +
                ↪  " is not a valid matrix kind");
199         }
200     }
201 }
```

Генератор профильных матриц

```java
package matrix;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.function.BiFunction;
import java.util.function.Function;

import static java.lang.Math.max;

public class ProfileMatrixGenerator extends Generator {
    ProfileMatrixGenerator(Args args) {
        super(args);
    }

    @Override
    protected void generate(BiFunction<Integer, Integer, Double> aValueGenerator,
                            Function<Integer, Double> bValueGenerator) throws
                            ↪ IOException {
        List<Double> di = new ArrayList<>(), al = new ArrayList<>(), au = new
        ↪ ArrayList<>(), b = new ArrayList<>();
        List<Integer> ia = new ArrayList<>();
        ia.add(1);
        for (int i = 1; i <= args.dim; i++) {
            ia.add(ia.get(ia.size() - 1));
            boolean flag = false;
            for (int j = max(1, i - args.belowDiagsCount); j < i; j++) {
                double dAL = valueOrZero(aValueGenerator.apply(i, j),
                ↪ args.zeroProbability), dAU;
                if (dAL == 0) {
                    dAU = 0;
                } else {
                    dAU = valueNonZero(aValueGenerator, i, j);
                }
                if (!flag) {
                    if (dAL != 0) {
                        flag = true;
                        al.add(dAL);
                        au.add(dAU);
                        ia.set(ia.size() - 1, ia.get(ia.size() - 1) + i - j);
                    }
                } else {
                    al.add(dAL);
                    au.add(dAU);
                }
            }
            di.add(valueNonZero(aValueGenerator, i, i));
            b.add(valueOrZero(bValueGenerator.apply(i), args.zeroProbability));
        }
        prepareDir();
        writeList(List.of(args.dim), "A");
        writeList(di, "A");
        writeList(al, "A");
        writeList(au, "A");
        writeList(ia, "A");
```

```
54          writeList(b, "b");
55      }
56
57      @Override
58      Matrix readMatrix(BufferedReader reader) throws IOException {
59          return new ProfileMatrix(reader);
60      }
61  }
```

Генератор разреженных матриц

```java
package matrix;

import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.function.BiFunction;
import java.util.function.Function;

import static java.lang.Math.max;

/**
 * Генератор разреженных матриц
 */
public class SparseMatrixGenerator extends Generator {
    SparseMatrixGenerator(Args args) {
        super(args);
    }

    @Override
    void generate(BiFunction<Integer, Integer, Double> aValueGenerator,
    ↪  Function<Integer, Double> bValueGenerator)
            throws IOException {
        List<Double> di = new ArrayList<>(), al = new ArrayList<>(), au = new
        ↪  ArrayList<>(), b = new ArrayList<>();
        List<Integer> ia = new ArrayList<>(), ja = new ArrayList<>();
        ia.add(1);
        for (int i = 1; i <= args.dim; i++) {
            ia.add(ia.get(ia.size() - 1));
            int temp = 0;
            for (int j = max(1, i - args.belowDiagsCount); j < i; j++) {
                double dAL = valueOrZero(aValueGenerator.apply(i, j),
                ↪  args.zeroProbability), dAU;
                if (dAL == 0) {
                    continue;
                }
                temp++;
//                dAU = valueNonZero(aValueGenerator, i, j);
                al.add(dAL);
                au.add(dAL);
                ja.add(j);
            }
            ia.set(ia.size() - 1, ia.get(ia.size() - 1) + temp);
            di.add(valueNonZero(aValueGenerator, i, i));
            b.add(valueOrZero(bValueGenerator.apply(i), args.zeroProbability));
        }
        prepareDir();
        writeList(List.of(args.dim), "A");
        writeList(di, "A");
        writeList(al, "A");
        writeList(au, "A");
        writeList(ia, "A");
        writeList(ja, "A");
        writeList(b, "b");
    }

```

```
54      @Override
55      Matrix readMatrix(BufferedReader reader) throws IOException {
56          return new SparseMatrix(reader);
57      }
58  }
```

```java
package matrix;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.Reader;
import java.util.Arrays;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

public interface Utils {
    static <T> List<T> parseList(BufferedReader reader, Function<String, T> parser)
    ↪  throws IOException {
        return Arrays.stream(reader.readLine().split("\\s+"))
                .map(parser)
                .collect(Collectors.toList());
    }

    static List<Double> parseDoubleList(BufferedReader reader) throws IOException {
        return parseList(reader, Double::parseDouble);
    }

    static List<Integer> parseIntegerList(BufferedReader reader) throws IOException {
        return parseList(reader, Integer::parseInt);
    }

    static boolean eof(Reader r) throws IOException {
        r.mark(1);
        int i = r.read();
        r.reset();
        return i < 0;
    }
}
```