МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РФ

ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Отчет

По лабораторной работе №3 «Итерационные методы решения систем линейных алгебраических уравнений»



Пермь 2022

**Задание**

1) Решить систему линейных алгебраических уравнений прямым методом квадратного корня.



2) Решить систему линейных алгебраических уравнений итерационными методами с критерием остановки (т.е. уменьшение начальной погрешности в 104 раз):



- методом простой итерации;

- градиентным методом наискорейшего спуска;

- методом ПВР;

- методом сопряженных градиентов.

В качества начального приближения выбирать вектор , который во всех вариантах принять равным .



Во всех случаях в качестве нормы вектора использовать энергетическую норму.

3) Для каждого метода получить число итераций, необходимое для достижения требуемой точности (по относительной норме погрешности), выдавая (на печать) на каждом шаге (или через заданное число шагов)

- номер итерации;

- значение параметров итерационного метода;

- оценку нормы матрицы перехода q (для метода простой итерации и градиентного метода наискорейшего спуска);

- значение нормы невязки;

- значение нормы погрешности;

- оценку погрешности приближенного решения ((для метода простой итерации и градиентного метода наискорейшего спуска);

- значение компонентов вектора приближенного решения.

4) Оценку нормы матрицы перехода осуществлять по формуле.



5) В методе простой итерации значения итерационного параметра вычислять по формуле



6) В методе ПВР получить решение при оптимальном значении параметра ω, которое необходимо определить, варьируя параметр ω в диапазоне (0, 2) с шагом 0.1, и производя вычисления с критерием остановки (или по критерию минимальности нормы вектора невязки при заданном числе итераций).



7) Провести анализ эффективности рассматриваемых методов:

- сравнив решение, полученное итерационным методом, с решением полученным прямым методом;

- сравнив фактическое число итераций, необходимое для достижения заданной точности, с теоретической оценкой, вычислив число обусловленности в эвклидовой норме.

**Исходные данные**

Матрица А для варианта №23:

Матрица А для варианта №1:

**Теоретическая справка**

Дано:



Задача:



1. **Прямой метод квадратного корня.**

Пусть A= AT – симметричная матрица

Будем строить разложение



где *S* – верхняя треугольная матрица, *D* – диагональная матрица.



Но



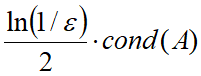
1. **Метод простой итерации.**



Расчетные формулы:



Теоретическая оценка:



По условию: значение итерационного параметра вычислять по формуле

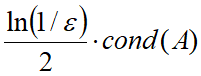


1. **Градиентным методом наискорейшего спуска.**

Расчетные формулы:



Теоретическая оценка:



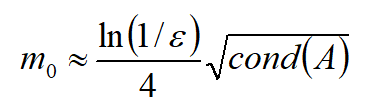
1. **Метод ПВР.**



Расчетные формулы:

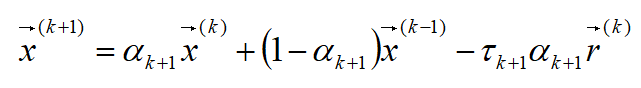


Теоретическая оценка:



1. **Методом сопряженных градиентов.**

Расчетные формулы:

,



Теоретическая оценка:



1. **Число обусловленности матрицы А.**



1. **Оценка нормы матрицы перехода.**



1. **Энергетическая норма вектора.**

Пусть A=AT>0, назовем энергетической нормой норму



**9. Норма матрицы.**

Евклидова норма.



**Решение**

Вариант №1.

Variant =

b

1.0000000 2.0000000 3.0000000 4.0000000

A

9.00000 2.50000 -1.70000 -3.00000

2.50000 8.00000 0.30000 3.80000

-1.70000 0.30000 8.00000 -1.00000

-3.00000 3.80000 -1.00000 8.30000

Решение прямым методом квадратного корня

x\*

0.7379696 -0.5125037 0.6842555 1.0657444

Норма матрицы = 12.1120398

Метод простой итерации

Iter | tau | q | Норма невязки | Норма погрешности | Оценка погрешности | X[1] X[2] X[3] X[4]

1 | 0.1486 |23.881 | 160.6958618 | 10.0186988 | -24.9251105 | 1.60931 -2.84477 0.63706 -0.57726

2 | 0.1486 | 0.746 | 119.9126714 | 7.9139777 | 52.3967288 | 0.56595 0.53429 0.77312 3.14791

3 | 0.1486 | 0.795 | 95.3009007 | 6.3110961 | 54.8411264 | 1.35787 -1.82615 0.88678 -0.07504

4 | 0.1486 | 0.798 | 76.0817699 | 5.0381897 | 44.7592381 | 0.55937 0.14054 0.69164 2.38043

5 | 0.1486 | 0.799 | 60.7780963 | 4.0238414 | 35.8718443 | 1.14363 -1.31227 0.80400 0.31147

6 | 0.1486 | 0.799 | 48.5691045 | 3.2147917 | 28.7141262 | 0.59216 -0.09152 0.68768 1.89216

7 | 0.1486 | 0.799 | 38.8223809 | 2.5690910 | 22.9805754 | 1.00009 -1.00471 0.75082 0.57055

8 | 0.1486 | 0.799 | 31.0376093 | 2.0535107 | 18.3901751 | 0.62841 -0.24023 0.68626 1.58608

9 | 0.1486 | 0.800 | 24.8175681 | 1.6416727 | 14.7156821 | 0.90628 -0.81717 0.72139 0.74195

10 | 0.1486 | 0.800 | 19.8463315 | 1.3126033 | 11.7747382 | 0.65938 -0.33628 0.68523 1.39395

11 | 0.1486 | 0.800 | 15.8723005 | 1.0496056 | 9.4211456 | 0.84560 -0.70198 0.70514 0.85470

12 | 0.1486 | 0.800 | 12.6949003 | 0.8393737 | 7.5377529 | 0.68323 -0.39845 0.68459 1.27311

13 | 0.1486 | 0.800 | 10.1541058 | 0.6712957 | 6.0307182 | 0.80661 -0.63083 0.69609 0.92856

14 | 0.1486 | 0.800 | 8.1221645 | 0.5369031 | 4.8248924 | 0.70059 -0.43871 0.68425 1.19696

15 | 0.1486 | 0.800 | 6.4970400 | 0.4294346 | 3.8601096 | 0.78166 -0.58666 0.69102 0.97677

16 | 0.1486 | 0.800 | 5.1972047 | 0.3434895 | 3.0882081 | 0.71281 -0.46479 0.68410 1.14888

17 | 0.1486 | 0.800 | 4.1574998 | 0.2747530 | 2.4706406 | 0.76576 -0.55911 0.68816 1.00815

18 | 0.1486 | 0.800 | 3.3258366 | 0.2197767 | 1.9765580 | 0.72121 -0.48167 0.68406 1.11848

19 | 0.1486 | 0.800 | 2.6605683 | 0.1758042 | 1.5812744 | 0.75563 -0.54188 0.68652 1.02852

20 | 0.1486 | 0.800 | 2.1283922 | 0.1406318 | 1.2650367 | 0.72690 -0.49260 0.68407 1.09924

21 | 0.1486 | 0.800 | 1.7026751 | 0.1124977 | 1.0120399 | 0.74919 -0.53106 0.68558 1.04172

22 | 0.1486 | 0.800 | 1.3621161 | 0.0899929 | 0.8096384 | 0.73070 -0.49966 0.68410 1.08703

23 | 0.1486 | 0.800 | 1.0896780 | 0.0719908 | 0.6477147 | 0.74510 -0.52424 0.68504 1.05026

24 | 0.1486 | 0.800 | 0.8717331 | 0.0575903 | 0.5181742 | 0.73322 -0.50422 0.68413 1.07929

25 | 0.1486 | 0.800 | 0.6973808 | 0.0460706 | 0.4145408 | 0.74250 -0.51994 0.68472 1.05577

26 | 0.1486 | 0.800 | 0.5579011 | 0.0368553 | 0.3316336 | 0.73487 -0.50717 0.68416 1.07437

27 | 0.1486 | 0.800 | 0.4463186 | 0.0294835 | 0.2653074 | 0.74085 -0.51723 0.68453 1.05933

28 | 0.1486 | 0.800 | 0.3570535 | 0.0235863 | 0.2122463 | 0.73596 -0.50907 0.68419 1.07124

29 | 0.1486 | 0.800 | 0.2856419 | 0.0188687 | 0.1697972 | 0.73980 -0.51550 0.68442 1.06162

30 | 0.1486 | 0.800 | 0.2285130 | 0.0150947 | 0.1358379 | 0.73667 -0.51030 0.68421 1.06925

31 | 0.1486 | 0.800 | 0.1828101 | 0.0120756 | 0.1086704 | 0.73914 -0.51441 0.68436 1.06309

32 | 0.1486 | 0.800 | 0.1462478 | 0.0096604 | 0.0869364 | 0.73713 -0.51108 0.68422 1.06798

33 | 0.1486 | 0.800 | 0.1169981 | 0.0077282 | 0.0695491 | 0.73871 -0.51372 0.68432 1.06404

34 | 0.1486 | 0.800 | 0.0935984 | 0.0061825 | 0.0556393 | 0.73743 -0.51159 0.68423 1.06717

35 | 0.1486 | 0.800 | 0.0748787 | 0.0049460 | 0.0445115 | 0.73844 -0.51328 0.68429 1.06465

36 | 0.1486 | 0.800 | 0.0599029 | 0.0039567 | 0.0356092 | 0.73762 -0.51192 0.68424 1.06665

37 | 0.1486 | 0.800 | 0.0479223 | 0.0031654 | 0.0284873 | 0.73827 -0.51300 0.68428 1.06504

38 | 0.1486 | 0.800 | 0.0383378 | 0.0025323 | 0.0227899 | 0.73774 -0.51213 0.68425 1.06632

39 | 0.1486 | 0.800 | 0.0306703 | 0.0020258 | 0.0182319 | 0.73816 -0.51282 0.68427 1.06529

40 | 0.1486 | 0.800 | 0.0245362 | 0.0016207 | 0.0145855 | 0.73782 -0.51226 0.68425 1.06612

41 | 0.1486 | 0.800 | 0.0196290 | 0.0012965 | 0.0116684 | 0.73809 -0.51270 0.68426 1.06546

Сравнение полученного решения с решением, полученным прямым методом

x

0.7380924 -0.5127045 0.6842645 1.0654557

x\*

0.7379696 -0.5125037 0.6842555 1.0657444

||x - x\*||

0.0001228 -0.0002008 0.0000090 -0.0002887

Метод наискорейшего спуска

Iter | tau | q | Норма невязки | Норма погрешности | Оценка погрешности | X[1] X[2] X[3] X[4]

1 | 0.0883 |14.195 | 160.6958618 | 2.5575154 | -15.2704289 | 1.36216 -0.87963 1.59552 1.27936

2 | 0.1178 | 0.166 | 19.9820321 | 0.9990155 | 0.4681340 | 1.06644 -0.84530 0.89975 1.56283

3 | 0.1390 | 0.282 | 4.7701328 | 0.7471003 | 0.2601110 | 1.02937 -0.86093 0.82069 1.33199

4 | 0.1515 | 0.742 | 3.2477338 | 0.5620614 | 1.4162744 | 0.92012 -0.70843 0.78654 1.35090

5 | 0.1385 | 0.752 | 2.6723437 | 0.4229140 | 1.1236662 | 0.90347 -0.70874 0.76373 1.21604

6 | 0.1515 | 0.752 | 1.8384133 | 0.3182190 | 0.8465594 | 0.84090 -0.62371 0.74172 1.22729

7 | 0.1385 | 0.752 | 1.5129753 | 0.2394421 | 0.6370628 | 0.83175 -0.62354 0.72928 1.15082

8 | 0.1515 | 0.752 | 1.0408609 | 0.1801670 | 0.4793566 | 0.79619 -0.57550 0.71680 1.15721

9 | 0.1385 | 0.752 | 0.8566073 | 0.1355659 | 0.3606910 | 0.79110 -0.57535 0.70973 1.11391

10 | 0.1515 | 0.752 | 0.5893091 | 0.1020059 | 0.2714006 | 0.77091 -0.54819 0.70269 1.11753

11 | 0.1385 | 0.752 | 0.4849895 | 0.0767540 | 0.2042147 | 0.76806 -0.54808 0.69868 1.09301

12 | 0.1515 | 0.752 | 0.3336521 | 0.0577532 | 0.1536606 | 0.75661 -0.53271 0.69470 1.09507

13 | 0.1385 | 0.752 | 0.2745890 | 0.0434562 | 0.1156215 | 0.75501 -0.53264 0.69242 1.08118

14 | 0.1515 | 0.752 | 0.1889057 | 0.0326984 | 0.0869990 | 0.74852 -0.52395 0.69017 1.08235

15 | 0.1385 | 0.752 | 0.1554656 | 0.0246038 | 0.0654621 | 0.74762 -0.52390 0.68888 1.07449

16 | 0.1515 | 0.752 | 0.1069538 | 0.0185131 | 0.0492567 | 0.74394 -0.51898 0.68760 1.07514

17 | 0.1385 | 0.752 | 0.0880208 | 0.0139301 | 0.0370631 | 0.74344 -0.51896 0.68687 1.07069

18 | 0.1515 | 0.752 | 0.0605546 | 0.0104816 | 0.0278880 | 0.74135 -0.51617 0.68615 1.07107

19 | 0.1385 | 0.752 | 0.0498353 | 0.0078869 | 0.0209842 | 0.74106 -0.51616 0.68574 1.06855

20 | 0.1515 | 0.752 | 0.0342846 | 0.0059345 | 0.0157895 | 0.73988 -0.51458 0.68533 1.06876

21 | 0.1385 | 0.752 | 0.0282155 | 0.0044654 | 0.0118808 | 0.73972 -0.51457 0.68509 1.06733

22 | 0.1515 | 0.752 | 0.0194111 | 0.0033599 | 0.0089396 | 0.73905 -0.51368 0.68486 1.06745

23 | 0.1385 | 0.752 | 0.0159749 | 0.0025282 | 0.0067266 | 0.73896 -0.51367 0.68473 1.06664

24 | 0.1515 | 0.752 | 0.0109901 | 0.0019023 | 0.0050614 | 0.73858 -0.51317 0.68460 1.06671

25 | 0.1385 | 0.752 | 0.0090446 | 0.0014314 | 0.0038084 | 0.73853 -0.51317 0.68452 1.06625

Сравнение полученного решения с решением, полученным прямым методом

x

0.7385314 -0.5131667 0.6845241 1.0662529

x\*

0.7379696 -0.5125037 0.6842555 1.0657444

||x - x\*||

0.0005618 -0.0006630 0.0002686 0.0005085

Метод ПВР - выбор оптимального w

w = 0.1000000 Iter = 111

w = 0.2000000 Iter = 55

w = 0.3000000 Iter = 36

w = 0.4000000 Iter = 27

w = 0.5000000 Iter = 21

w = 0.6000000 Iter = 17

w = 0.7000000 Iter = 14

w = 0.8000000 Iter = 12

w = 0.9000000 Iter = 10

w = 1.0000000 Iter = 8

w = 1.1000000 Iter = 7

w = 1.2000000 Iter = 6

w = 1.3000000 Iter = 5

w = 1.4000000 Iter = 6

w = 1.5000000 Iter = 8

w = 1.6000000 Iter = 10

w = 1.7000000 Iter = 14

w = 1.8000000 Iter = 22

w = 1.9000000 Iter = 47

w\* = 1.3000000 IterMin = 5

Метод ПВР

Iter | w | Норма невязки | Норма погрешности | X[1] X[2] X[3] X[4]

1 |1.3000000 | 160.6958618 | 6.5610052 | 1.59222 -3.53809 0.84983 2.41357

2 | 1.30000 | 56.6508181 | 3.4067660 | 2.19898 -1.03872 1.28286 1.75485

3 | 1.30000 | 29.3986952 | 0.8730316 | 0.93529 -0.88950 0.68954 1.17694

4 | 1.30000 | 8.1144922 | 0.3401711 | 0.86440 -0.51969 0.73601 1.10417

5 | 1.30000 | 3.3426407 | 0.0763740 | 0.73200 -0.53417 0.67438 1.06276

6 | 1.30000 | 0.7916239 | 0.0298545 | 0.74387 -0.50608 0.68805 1.06618

7 | 1.30000 | 0.3072711 | 0.0106712 | 0.73500 -0.51368 0.68242 1.06463

8 | 1.30000 | 0.1059966 | 0.0034545 | 0.73835 -0.51153 0.68468 1.06575

9 | 1.30000 | 0.0352105 | 0.0013375 | 0.73761 -0.51267 0.68404 1.06564

Сравнение полученного решения с решением, полученным прямым методом

x

0.7376084 -0.5126708 0.6840361 1.0656393

x\*

0.7379696 -0.5125037 0.6842555 1.0657444

||x - x\*||

-0.0003612 -0.0001672 -0.0002194 -0.0001051

Метод сопряженный градиентов

Iter | tau | Норма невязки | Норма погрешности | X[1] X[2] X[3] X[4] Alpha

1 |0.0883 | 160.6958618 | 2.5575154 | 1.36216 -0.87963 1.59552 1.27936 | 1.00000000000000

2 |0.1178 | 19.9820321 | 0.9171896 | 1.06832 -0.92578 0.84034 1.49389 | 1.02828780603163

3 |0.4476 | 1.8428276 | 0.2622362 | 0.83021 -0.52211 0.66940 1.09809 | 1.13552922585610

4 |0.0888 | 2.8288748 | 0.0000000 | 0.73797 -0.51250 0.68426 1.06574 | 1.08902342540090

Сравнение полученного решения с решением, полученным прямым методом

x

0.7379696 -0.5125037 0.6842555 1.0657444

x\*

0.7379696 -0.5125037 0.6842555 1.0657444

||x - x\*||

0.0000000 -0.0000000 0.0000000 0.0000000

Число обусловленности = 7.098

Теоретическая оценка числа итераций

Методы простой итерации и градиентного спуска: 33

Метод релаксации: 6

Метод сопряженных градиентов: 13

Фактическое число итераций

Методы простой итерации: 41

Метод градиентного спуска: 25

Метод релаксации: 9

Метод сопряженных градиентов: 4

Вариант №23.

Variant =

b

1.0000000 2.0000000 3.0000000 4.0000000

A

8.50000 3.60000 0.80000 -4.10000

3.60000 9.90000 2.10000 -4.70000

0.80000 2.10000 5.40000 -2.60000

-4.10000 -4.70000 -2.60000 6.90000

Решение прямым методом квадратного корня

x\*

0.6125315 0.5784715 1.0798453 1.7446077

Норма матрицы = 17.6320712

Метод простой итерации

Iter | tau | q | Норма невязки | Норма погрешности | Оценка погрешности | X[1] X[2] X[3] X[4]

1 | 0.1021 | 3.536 | 34.6356608 | 3.9768079 | -4.9301870 | 0.92854 1.09143 2.20372 3.76520

2 | 0.1021 | 0.722 | 25.0015780 | 2.9384449 | 6.6235930 | 1.21975 1.18542 1.98471 3.01859

3 | 0.1021 | 0.774 | 19.3455787 | 2.2182422 | 6.7549677 | 0.92912 0.76612 1.64432 2.90675

4 | 0.1021 | 0.790 | 15.2762638 | 1.6857153 | 5.8543971 | 1.02576 0.89671 1.57552 2.46051

5 | 0.1021 | 0.794 | 12.1312123 | 1.2841966 | 4.7769403 | 0.80939 0.66044 1.39032 2.41345

6 | 0.1021 | 0.796 | 9.6512165 | 0.9796551 | 3.8342669 | 0.86303 0.75960 1.36304 2.14645

7 | 0.1021 | 0.796 | 7.6845733 | 0.7481684 | 3.0653715 | 0.72416 0.61657 1.25429 2.13032

8 | 0.1021 | 0.797 | 6.1220173 | 0.5720070 | 2.4486274 | 0.76049 0.68471 1.24322 1.96993

9 | 0.1021 | 0.797 | 4.8793450 | 0.4378234 | 1.9558503 | 0.67403 0.59605 1.17810 1.96748

10 | 0.1021 | 0.797 | 3.8904198 | 0.3355210 | 1.5624232 | 0.69946 0.64155 1.17430 1.87074

11 | 0.1021 | 0.798 | 3.1029984 | 0.2574500 | 1.2483176 | 0.64593 0.58612 1.13509 1.87362

12 | 0.1021 | 0.798 | 2.4757208 | 0.1978097 | 0.9974995 | 0.66362 0.61617 1.13451 1.81505

13 | 0.1021 | 0.798 | 1.9758015 | 0.1521990 | 0.7971790 | 0.63045 0.58137 1.11082 1.81942

14 | 0.1021 | 0.798 | 1.5772274 | 0.1172763 | 0.6371604 | 0.64262 0.60111 1.11151 1.78384

15 | 0.1021 | 0.798 | 1.2593419 | 0.0905037 | 0.5093143 | 0.62202 0.57921 1.09716 1.78806

16 | 0.1021 | 0.799 | 1.0057299 | 0.0699519 | 0.4071572 | 0.63029 0.59211 1.09822 1.76637

17 | 0.1021 | 0.799 | 0.8033382 | 0.0541533 | 0.3255167 | 0.61748 0.57830 1.08949 1.76990

18 | 0.1021 | 0.799 | 0.6417807 | 0.0419909 | 0.2602647 | 0.62304 0.58672 1.09052 1.75663

19 | 0.1021 | 0.799 | 0.5127890 | 0.0326134 | 0.2081060 | 0.61505 0.57800 1.08520 1.75935

20 | 0.1021 | 0.799 | 0.4097773 | 0.0253718 | 0.1664096 | 0.61877 0.58348 1.08606 1.75122

21 | 0.1021 | 0.799 | 0.3274977 | 0.0197706 | 0.1330742 | 0.61378 0.57796 1.08280 1.75322

22 | 0.1021 | 0.799 | 0.2617668 | 0.0154309 | 0.1064213 | 0.61625 0.58152 1.08347 1.74822

23 | 0.1021 | 0.799 | 0.2092482 | 0.0120631 | 0.0851099 | 0.61312 0.57803 1.08147 1.74965

24 | 0.1021 | 0.799 | 0.1672807 | 0.0094450 | 0.0680686 | 0.61475 0.58033 1.08196 1.74656

25 | 0.1021 | 0.799 | 0.1337404 | 0.0074064 | 0.0544412 | 0.61279 0.57812 1.08073 1.74757

26 | 0.1021 | 0.800 | 0.1069322 | 0.0058163 | 0.0435431 | 0.61386 0.57961 1.08108 1.74566

27 | 0.1021 | 0.800 | 0.0855029 | 0.0045741 | 0.0348275 | 0.61263 0.57821 1.08033 1.74635

28 | 0.1021 | 0.800 | 0.0683717 | 0.0036020 | 0.0278571 | 0.61333 0.57917 1.08057 1.74516

29 | 0.1021 | 0.800 | 0.0546756 | 0.0028401 | 0.0222821 | 0.61255 0.57828 1.08010 1.74564

30 | 0.1021 | 0.800 | 0.0437249 | 0.0022421 | 0.0178232 | 0.61301 0.57890 1.08027 1.74490

31 | 0.1021 | 0.800 | 0.0349688 | 0.0017720 | 0.0142567 | 0.61252 0.57834 1.07998 1.74522

32 | 0.1021 | 0.800 | 0.0279671 | 0.0014019 | 0.0114041 | 0.61282 0.57874 1.08010 1.74476

33 | 0.1021 | 0.800 | 0.0223680 | 0.0011103 | 0.0091224 | 0.61251 0.57838 1.07992 1.74497

34 | 0.1021 | 0.800 | 0.0178904 | 0.0008801 | 0.0072972 | 0.61271 0.57864 1.07999 1.74468

35 | 0.1021 | 0.800 | 0.0143094 | 0.0006982 | 0.0058373 | 0.61251 0.57841 1.07988 1.74482

36 | 0.1021 | 0.800 | 0.0114455 | 0.0005544 | 0.0046695 | 0.61264 0.57857 1.07993 1.74464

Сравнение полученного решения с решением, полученным прямым методом

x

0.6126394 0.5785743 1.0799340 1.7446426

x\*

0.6125315 0.5784715 1.0798453 1.7446077

||x - x\*||

0.0001079 0.0001029 0.0000887 0.0000349

Метод наискорейшего спуска

Iter | tau | q | Норма невязки | Норма погрешности | Оценка погрешности | X[1] X[2] X[3] X[4]

1 | 0.1216 | 4.210 | 34.6356608 | 3.9191797 | -5.5218880 | 0.91491 0.91809 2.05181 3.72041

2 | 0.0799 | 0.636 | 33.5340858 | 2.8615980 | 4.6794731 | 1.19683 1.14124 1.96663 3.06000

3 | 0.1295 | 0.720 | 14.8844912 | 2.1153297 | 4.9474916 | 0.89789 0.70687 1.57588 2.83601

4 | 0.0809 | 0.734 | 17.4670928 | 1.5735848 | 3.8915143 | 0.99421 0.85170 1.54846 2.47444

5 | 0.1301 | 0.743 | 8.0741979 | 1.1713541 | 3.0423956 | 0.78463 0.63920 1.35168 2.34842

6 | 0.0810 | 0.744 | 9.6533206 | 0.8722323 | 2.2731261 | 0.83134 0.72394 1.33846 2.14849

7 | 0.1302 | 0.745 | 4.4721488 | 0.6495224 | 1.6972236 | 0.70969 0.61034 1.23081 2.07903

8 | 0.0810 | 0.745 | 5.3521478 | 0.4836889 | 1.2641441 | 0.73478 0.65809 1.22349 1.96832

9 | 0.1302 | 0.745 | 2.4798538 | 0.3601969 | 0.9415570 | 0.66664 0.59579 1.16374 1.92995

10 | 0.0810 | 0.745 | 2.9680307 | 0.2682347 | 0.7011796 | 0.68045 0.62242 1.15962 1.86860

11 | 0.1302 | 0.745 | 1.3752194 | 0.1997516 | 0.5221709 | 0.64257 0.58800 1.12642 1.84737

12 | 0.0810 | 0.745 | 1.6459543 | 0.1487530 | 0.3888563 | 0.65022 0.60280 1.12412 1.81335

13 | 0.1302 | 0.745 | 0.7626446 | 0.1107748 | 0.2895782 | 0.62920 0.58374 1.10569 1.80159

14 | 0.0810 | 0.745 | 0.9127851 | 0.0824929 | 0.2156461 | 0.63343 0.59195 1.10441 1.78273

15 | 0.1302 | 0.745 | 0.4229345 | 0.0614317 | 0.1605896 | 0.62177 0.58139 1.09418 1.77620

16 | 0.0810 | 0.745 | 0.5061970 | 0.0457475 | 0.1195895 | 0.62412 0.58594 1.09347 1.76575

17 | 0.1302 | 0.745 | 0.2345439 | 0.0340677 | 0.0890571 | 0.61766 0.58009 1.08780 1.76213

18 | 0.0810 | 0.745 | 0.2807182 | 0.0253699 | 0.0663199 | 0.61896 0.58262 1.08740 1.75633

19 | 0.1302 | 0.745 | 0.1300694 | 0.0188927 | 0.0493878 | 0.61537 0.57937 1.08425 1.75432

20 | 0.0810 | 0.745 | 0.1556760 | 0.0140692 | 0.0367786 | 0.61610 0.58077 1.08404 1.75111

21 | 0.1302 | 0.745 | 0.0721317 | 0.0104772 | 0.0273887 | 0.61411 0.57897 1.08229 1.75000

22 | 0.0810 | 0.745 | 0.0863322 | 0.0078023 | 0.0203961 | 0.61451 0.57975 1.08217 1.74821

23 | 0.1302 | 0.745 | 0.0400016 | 0.0058103 | 0.0151887 | 0.61341 0.57875 1.08120 1.74760

24 | 0.0810 | 0.745 | 0.0478767 | 0.0043269 | 0.0113109 | 0.61363 0.57918 1.08113 1.74661

25 | 0.1302 | 0.745 | 0.0221834 | 0.0032222 | 0.0084231 | 0.61302 0.57862 1.08060 1.74626

26 | 0.0810 | 0.745 | 0.0265506 | 0.0023995 | 0.0062726 | 0.61314 0.57886 1.08056 1.74572

27 | 0.1302 | 0.745 | 0.0123021 | 0.0017869 | 0.0046712 | 0.61280 0.57856 1.08026 1.74553

28 | 0.0810 | 0.745 | 0.0147240 | 0.0013307 | 0.0034786 | 0.61287 0.57869 1.08024 1.74522

29 | 0.1302 | 0.745 | 0.0068223 | 0.0009909 | 0.0025904 | 0.61268 0.57852 1.08008 1.74512

30 | 0.0810 | 0.745 | 0.0081654 | 0.0007379 | 0.0019291 | 0.61272 0.57859 1.08007 1.74495

31 | 0.1302 | 0.745 | 0.0037834 | 0.0005495 | 0.0014366 | 0.61261 0.57850 1.07997 1.74489

Сравнение полученного решения с решением, полученным прямым методом

x

0.6126142 0.5784975 1.0799736 1.7448903

x\*

0.6125315 0.5784715 1.0798453 1.7446077

||x - x\*||

0.0000827 0.0000261 0.0001283 0.0002826

Метод ПВР - выбор оптимального w

w = 0.1000000 Iter = 123

w = 0.2000000 Iter = 58

w = 0.3000000 Iter = 36

w = 0.4000000 Iter = 26

w = 0.5000000 Iter = 19

w = 0.6000000 Iter = 15

w = 0.7000000 Iter = 11

w = 0.8000000 Iter = 9

w = 0.9000000 Iter = 7

w = 1.0000000 Iter = 5

w = 1.1000000 Iter = 4

w = 1.2000000 Iter = 5

w = 1.3000000 Iter = 6

w = 1.4000000 Iter = 7

w = 1.5000000 Iter = 9

w = 1.6000000 Iter = 13

w = 1.7000000 Iter = 17

w = 1.8000000 Iter = 28

w = 1.9000000 Iter = 58

w\* = 1.1000000 IterMin = 4

Метод ПВР

Iter | w | Норма невязки | Норма погрешности | X[1] X[2] X[3] X[4]

1 |1.1000000 | 34.6356608 | 2.1888962 | 0.90941 1.04735 1.83340 2.37677

2 | 1.10000 | 23.0971791 | 0.5504430 | 0.62181 0.68218 1.29343 1.85369

3 | 1.10000 | 5.3722750 | 0.0942601 | 0.59905 0.58062 1.11754 1.74212

4 | 1.10000 | 0.6446626 | 0.0261058 | 0.60766 0.57011 1.07913 1.73511

5 | 1.10000 | 0.2282071 | 0.0123632 | 0.61195 0.57475 1.07658 1.74103

6 | 1.10000 | 0.1237138 | 0.0036235 | 0.61277 0.57765 1.07859 1.74398

7 | 1.10000 | 0.0330600 | 0.0006764 | 0.61269 0.57846 1.07962 1.74467

8 | 1.10000 | 0.0041083 | 0.0002228 | 0.61258 0.57854 1.07986 1.74469

Сравнение полученного решения с решением, полученным прямым методом

x

0.6125786 0.5785384 1.0798635 1.7446902

x\*

0.6125315 0.5784715 1.0798453 1.7446077

||x - x\*||

0.0000471 0.0000670 0.0000182 0.0000825

Метод сопряженный градиентов

Iter | tau | Норма невязки | Норма погрешности | X[1] X[2] X[3] X[4] Alpha

1 |0.1216 | 34.6356608 | 3.9191797 | 0.91491 0.91809 2.05181 3.72041 | 1.00000000000000

2 |0.0799 | 33.5340858 | 1.8212848 | 1.33054 0.55785 1.26463 2.42142 | 1.67933495441832

3 |0.1910 | 8.4367274 | 0.0699668 | 0.61625 0.55920 1.10434 1.73712 | 1.27503222401780

4 |0.2120 | 0.3297755 | 0.0000000 | 0.61253 0.57847 1.07985 1.74461 | 1.00147798427012

Сравнение полученного решения с решением, полученным прямым методом

x

0.6125315 0.5784715 1.0798453 1.7446077

x\*

0.6125315 0.5784715 1.0798453 1.7446077

||x - x\*||

0.0000000 0.0000000 0.0000000 -0.0000000

Число обусловленности = 7.359

Теоретическая оценка числа итераций

Методы простой итерации и градиентного спуска: 34

Метод релаксации: 6

Метод сопряженных градиентов: 13

Фактическое число итераций

Методы простой итерации: 36

Метод градиентного спуска: 31

Метод релаксации: 8

Метод сопряженных градиентов: 4

**Вывод**

Каждый из рассмотренных методов позволяет достаточно точно решить СЛАУ. Но все эти методы имеют свои достоинства и недостатки.

**Код программы**

#include <iostream>

#include <iomanip>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <fstream>

using namespace std;

ifstream input("C:/Users/1/source/repos/lab3/input.txt", ios::in);

ofstream output("C:/Users/1/source/repos/lab3/output.txt", ios::out);

const int n = 4;

const double eps = 0.0001;

const double eps1 = 0.01;

void MatrixOutput(double\*\* A) // вывод матрицы

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

output << fixed << setprecision(5) << setw(10) << A[i][j];

}

output << endl;

}

output << endl;

}

void OutputVect(double\* B) // вывод вектора

{

for (int i = 0; i < n; i++)

output << setw(12) << fixed << setprecision(7) << B[i];

output << endl;

}

double\* MultiMatrixVector(double\*\* A, double\* B) // умножение матрицы на вектор

{

double\* result = new double[n];

for (int i = 0; i < n; i++)

result[i] = 0;

for (int i = 0; i < n; ++i) {

double s = 0;

for (int j = 0; j < n; ++j)

s += A[i][j] \* B[j];

result[i] = s;

}

return result;

}

double\* SubtractVector(double\* A, double\* B) //разность векторов

{

double\* result = new double[n];

for (int i = 0; i < n; i++)

result[i] = 0;

for (int i = 0; i < n; ++i)

result[i] = A[i] - B[i];

return result;

}

double DotProductOfVector(double\* A, double\* B) //скалярное произведение векторов

{

double sum = 0;

for (int i = 0; i < n; ++i)

sum += A[i] \* B[i];

return sum;

}

double\* AddVector(double\* A, double\* B) //сложение векторов

{

double\* result = new double[n];

for (int i = 0; i < n; i++)

result[i] = 0;

for (int i = 0; i < n; ++i)

result[i] = A[i] + B[i];

return result;

}

double\* MultiVectorNum(double\* A, double B) //умножение вектора на число

{

double\* result = new double[n];

for (int i = 0; i < n; i++)

result[i] = 0;

for (int i = 0; i < n; ++i)

result[i] = A[i] \* B;

return result;

}

double\*\* MultiMatrix(double\*\* A, double\*\* B) //умножение матрицы на матрицу

{

double\*\* result = new double\* [n];

for (int i = 0; i < n; i++)

result[i] = new double[n];

double sum;

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

{

sum = 0;

for (int k = 0; k < n; ++k)

sum += A[i][k] \* B[k][j];

result[i][j] = sum;

}

return result;

}

double\* ResidualVector(double\*\* A, double\* x, double\* B) //вычисление вектора невязки Ax-b

{

return SubtractVector(MultiMatrixVector(A, x), B);

}

double VectorEnergyNorm(double\*\* A, double\* X) // энергетическая норма вектора

{

return sqrt(DotProductOfVector(MultiMatrixVector(A, X), X));

}

double qCalc(double\*\* A, double\* xPrev, double\* xCur, double\* xNext) //оценка нормы матрицы q

{

return VectorEnergyNorm(A, SubtractVector(xNext, xCur)) / VectorEnergyNorm(A, SubtractVector(xCur, xPrev));

}

double Error(double\*\* A, double\* xCur, double\* xNext, double q) // оценка погрешности приближенного решения

{

return q \* VectorEnergyNorm(A, SubtractVector(xNext, xCur)) / (1 - q);

}

double\*\* Jacobi\_Rotation(double\*\* A) // метод вращений Якоби

{

double\*\* result = new double\* [n];

for (int i = 0; i < n; i++)

result[i] = new double[n];

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

result[i][j] = A[i][j];

double theta, c, s;

double eps = 0.0000001;

while (true)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = i + 1; j < n; j++)

{

int t1 = 0, t2 = 1;

for (int c1 = 0; c1 <n - 1; c1++)

for (int c2 = c1 + 1; c2 < n; c2++)

if (abs(result[c1][c2]) > abs(result[t1][t2])) { t1 = c1; t2 = c2; }

if (abs(result[t1][t2]) < eps) return result;

if (result[t1][t1] == result[t2][t2]) theta = M\_PI / 4;

else theta = atan(2 \* result[t1][t2] / (result[t1][t1] - result[t2][t2])) / 2;

c = cos(theta);

s = sin(theta);

for (int m = 0; m < n; m++)

{

double qi = result[m][t1]; double qj = result[m][t2];

result[m][t1] = c \* qi + s \* qj;

result[m][t2] = -s \* qi + c \* qj;

}

for (int m = 0; m < n; m++)

{

double qi = result[t1][m]; double qj = result[t2][m];

result[t1][m] = c \* qi + s \* qj;

result[t2][m] = -s \* qi + c \* qj;

}

}

}

}

return result;

}

double MatrixNormCalc(double\*\* A) //вычисление нормы матрицы

{

double\*\* AT = new double\* [n];

for (int i = 0; i < n; i++)

AT[i] = new double[n];

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

AT[j][i] = A[i][j];

double\*\* result = new double\* [n];

for (int i = 0; i < n; i++)

result[i] = new double[n];

result = Jacobi\_Rotation(MultiMatrix(AT, A));

double max = result[0][0];

for (int i = 1; i < n; i++)

if (max < result[i][i])

max = result[i][i];

return sqrt(max);

}

double tauCalc(double\*\* A) //вычисление tau

{

return 0.9 \* (2 / MatrixNormCalc(A));

}

double\*\* InverseMatrix(double\*\* A) // обратная матрица

{

double temp;

double\*\* result = new double\* [n];

for (int i = 0; i < n; i++)

result[i] = new double[n];

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

if (i == j)

result[i][j] = 1;

else result[i][j] = 0;

for (int k = 0; k < n; k++) {

temp = A[k][k];

for (int j = 0; j < n; j++) {

A[k][j] /= temp;

result[k][j] /= temp;

}

for (int i = k + 1; i < n; i++) {

temp = A[i][k];

for (int j = 0; j < n; j++) {

A[i][j] -= A[k][j] \* temp;

result[i][j] -= result[k][j] \* temp;

}

}

}

for (int k = n - 1; k > 0; k--) {

for (int i = k - 1; i >= 0; i--) {

temp = A[i][k];

for (int j = 0; j < n; j++) {

A[i][j] -= A[k][j] \* temp;

result[i][j] -= result[k][j] \* temp;

}

}

}

return result;

}

double CondNumberCalc(double\*\* A) // число обусловленности матрицы

{

return MatrixNormCalc(A) \* MatrixNormCalc(InverseMatrix(A));

}

double\* MethodSquareRoot(double\*\* A, double\* B) //прямой метод квадратного корня

{

double\*\* U = new double\* [n];

for (int i = 0; i < n; i++)

U[i] = new double[n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

U[i][j] = 0;

double sum = 0;

double\* y = new double[n];

for (int i = 0; i < n; i++)

y[i] = 0;

double\* x = new double[n];

for (int i = 0; i < n; i++)

x[i] = 0;

//Находим матрицу U по формулам

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

{

sum = 0;

if (i == j)

{

for (int k = 0; k < n; k++)

sum += U[k][i] \* U[k][i];

U[i][i] = sqrt(A[i][i] - sum);

}

if (i < j)

{

for (int k = 0; k < i; k++)

sum += U[k][i] \* U[k][j];

U[i][j] = (A[i][j] - sum) / U[i][i];

}

if (i > j)

U[i][j] = 0;

}

double\*\* UT = new double\* [n];

for (int i = 0; i < n; i++)

UT[i] = new double[4];

//Транспонирование матрицы U

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

{

UT[i][j] = U[j][i];

}

//Находим y из UT\*y=b

for (int i = 0; i < n; i++)

{

sum = 0;

for (int j = 0; j < i; j++)

sum += (UT[i][j] \* y[j]);

y[i] = (B[i] - sum) / UT[i][i];

}

//Находим x из U\*x=y

for (int i = n - 1; i >= 0; i--)

{

sum = 0;

for (int j = n - 1; j > i; j--)

sum += U[i][j] \* x[j];

x[i] = (y[i] - sum) / U[i][i];

}

return x;

}

int SimpleIterMethod(double\*\* A, double\* B) // метод простой итерации

{

output << endl << "Метод простой итерации" << endl;

output << setw(5) << "Iter |" << setw(9) << "tau |" << setw(8) << "q |" << setw(16) << "Норма невязки |"

<< setw(20) << "Норма погрешности |" << setw(21) << "Оценка погрешности |"

<< setw(9) << "X[1]" << setw(10) << "X[2]" << setw(10) << "X[3]" << setw(10) << "X[4]" << endl;

double\* X\_prev = new double[n];

for (int i = 0; i < n; i++)

X\_prev[i] = 0;

double\* X\_cur = new double[n];

double\* X0 = new double[n];

for (int i = 0; i < n; i++)

{

X\_cur[i] = B[i];

X0[i] = B[i];

}

double\* X = new double[n];

double\* X\_q = new double[n];

X\_q = MethodSquareRoot(A, B);

double\* r = new double[n];

double nevNorm = 1, tau = tauCalc(A), q = 0, norm\_resid = 0, norm\_error = 0, estim\_error = 0;

int iter = 0;

do {

r = ResidualVector(A, X\_cur, B);

X = SubtractVector(X\_cur, MultiVectorNum(r, tau));

if (iter == 0)

q = VectorEnergyNorm(A, SubtractVector(X\_cur, X));

else

q = qCalc(A, X\_prev, X\_cur, X);

estim\_error = Error(A, X\_cur, X, q);

norm\_error = VectorEnergyNorm(A, SubtractVector(X, X\_q));

norm\_resid = VectorEnergyNorm(A, r);

output << setw(4) << iter + 1 << " |" << fixed << setprecision(4) << setw(7) << tau << " |" << setw(6) << fixed << setprecision(3) << q

<< " |" << setprecision(7) << setw(12) << norm\_resid << " |" << setw(14) << norm\_error << " |"

<< setw(15) << estim\_error << " |";

for (int i = 0; i < 4; i++)

output << setw(10) << fixed << setprecision(5) << X[i];

output << endl;

X\_prev = X\_cur;

X\_cur = X;

iter++;

} while (VectorEnergyNorm(A, SubtractVector(X, X\_q)) / VectorEnergyNorm(A, SubtractVector(X0, X\_q)) > eps);

output << endl << "Сравнение полученного решения с решением, полученным прямым методом" << endl << "x" << endl;

OutputVect(X);

output << endl << "x\*" << endl;

OutputVect(X\_q);

output << endl << "||x - x\*||" << endl;

OutputVect(SubtractVector(X, X\_q));

output << endl;

return iter;

}

int ConjugateGradientMethod(double\*\* A, double\* B) // метод сопряженных градиентов

{

output << endl<< "Метод сопряженный градиентов" << endl;

output << setw(5) << "Iter |" << setw(8) << "tau |" << setw(16) << "Норма невязки |"

<< setw(20) << "Норма погрешности |"

<< setw(9) << "X[1]" << setw(10) << "X[2]" << setw(10) << "X[3]" << setw(10) << "X[4]" << setw(13) << "Alpha" << endl;

double\* X\_q = new double[n];//x\*

double\* X = new double[n];

double\* X\_now = new double[n];

double\* X\_pred = new double[n];

double\* X0 = new double[n];

double\* r = new double[n];

for (int i = 0; i < n; i++)

{

X\_pred[i] = B[i];

X\_now[i] = B[i];

X0[i] = B[i];

X[i] = B[i];

}

X\_q = MethodSquareRoot(A, B);

int iter = 0;

double tau, tau\_old, alpha = 1, alpha\_old = 1, norm\_error, norm\_resid;

r = ResidualVector(A, X\_pred, B);

tau = DotProductOfVector(r, r) / DotProductOfVector(MultiMatrixVector(A, r), r);

do {

if (iter > 0) {

r = ResidualVector(A, X, B);

tau = DotProductOfVector(r, r) / DotProductOfVector(MultiMatrixVector(A, r), r);

alpha = 1 / (1 - (tau / tau\_old) \* (1 / alpha\_old) \* DotProductOfVector(r, r) /

DotProductOfVector(ResidualVector(A, X\_pred, B), ResidualVector(A, X\_pred, B)));

}

X\_now = SubtractVector(AddVector(MultiVectorNum(X, alpha), MultiVectorNum(X\_pred, 1 - alpha)), MultiVectorNum(r, alpha \* tau));

norm\_error = VectorEnergyNorm(A, SubtractVector(X\_now, X\_q));

norm\_resid = VectorEnergyNorm(A, r);

output << setw(3) << iter + 1 << " |" << fixed << setprecision(4) << tau

<< " |" << setprecision(7) << setw(12) << norm\_resid << " |" << setw(14) << norm\_error << " |";

for (int i = 0; i < n; i++)

output << setw(10) << fixed << setprecision(5) << X\_now[i];

output << fixed << setprecision(14) << " | " << alpha << endl;

alpha\_old = alpha;

tau\_old = tau;

iter++;

X\_pred = X;

X = X\_now;

} while (VectorEnergyNorm(A, SubtractVector(X, X\_q)) / VectorEnergyNorm(A, SubtractVector(X0, X\_q)) > eps);

output << endl << "Сравнение полученного решения с решением, полученным прямым методом" << endl << "x" << endl;

OutputVect(X);

output << endl << "x\*" << endl;

OutputVect(X\_q);

output << endl << "||x - x\*||" << endl;

OutputVect(SubtractVector(X, X\_q));

output << endl;

return iter;

}

int SteepestDescentGradientMethod(double\*\* A, double\* B) // метод наискорейшего спуска

{

output << endl << "Метод наискорейшего спуска" << endl;

output << setw(5) << "Iter |" << setw(9) << "tau |" << setw(8) << "q |" << setw(16) << "Норма невязки |"

<< setw(20) << "Норма погрешности |" << setw(21) << "Оценка погрешности |"

<< setw(9) << "X[1]" << setw(10) << "X[2]" << setw(10) << "X[3]" << setw(10) << "X[4]" << endl;

double\* X\_q = new double[n];//x\*

double\* X = new double[n];

double\* X\_prpr = new double[n];

double\* X\_pred = new double[n];

double\* X0 = new double[n];

double\* r = new double[n];

for (int i = 0; i < n; i++)

{

X\_pred[i] = B[i];

X\_prpr[i] = B[i];

X0[i] = B[i];

}

X\_q = MethodSquareRoot(A, B);

int iter = 0;

double t, q, estim\_error, norm\_error, norm\_resid, tau;

do {

r = ResidualVector(A, X\_pred, B);

tau = DotProductOfVector(r, r) / DotProductOfVector(MultiMatrixVector(A, r), r);

X = SubtractVector(X\_pred, MultiVectorNum(r, tau));

if (iter == 0)

q = VectorEnergyNorm(A, SubtractVector(X\_pred, X));

else

q = qCalc(A, X\_prpr, X\_pred, X);

estim\_error = Error(A, X\_pred, X, q);

norm\_error = VectorEnergyNorm(A, SubtractVector(X, X\_q));

norm\_resid = VectorEnergyNorm(A, r);

X\_prpr = X\_pred;

X\_pred = X;

output << setw(4) << iter + 1 << " |" << fixed << setprecision(4) << setw(7) << tau << " |" << setw(6) << fixed << setprecision(3) << q

<< " |" << setprecision(7) << setw(12) << norm\_resid << " |" << setw(14) << norm\_error << " |"

<< setw(15) << estim\_error << " |";

for (int i = 0; i < 4; i++)

output << setw(10) << fixed << setprecision(5) << X[i];

output << endl;

iter++;

} while (VectorEnergyNorm(A, SubtractVector(X, X\_q)) / VectorEnergyNorm(A, SubtractVector(X0, X\_q)) > eps);

output << endl << "Сравнение полученного решения с решением, полученным прямым методом" << endl << "x" << endl;

OutputVect(X);

output << endl << "x\*" << endl;

OutputVect(X\_q);

output << endl << "||x - x\*||" << endl;

OutputVect(SubtractVector(X, X\_q));

output << endl;

return iter;

}

int SOR(double\*\* A, double\* B) //ПВР

{

output << endl << "Метод ПВР - выбор оптимального w" << endl;

int iter = 0, iter\_best = -1;

double w = 0.1, w\_best = 3;

double\* X = new double[n];

double\* X0 = new double[n];

double\* r = new double[n];

double norm\_error, norm\_resid;

for (int i = 0; i < n; i++)

{

X0[i] = B[i];

}

double\* X\_q = new double[n];

X\_q = MethodSquareRoot(A, B);

while (w < 2)

{

iter = 0;

for (int i = 0; i < n; i++)

X[i] = B[i];

do {

for (int i = 0; i < n; i++) {

double sum = 0;

for (int j = 0; j < i; j++)

sum = sum + A[i][j] \* X[j];

for (int j = i + 1; j < n; j++)

sum = sum + A[i][j] \* X[j];

X[i] = (1 - w) \* X[i] + w \* (B[i] - sum) / A[i][i];

}

iter++;

} while (VectorEnergyNorm(A, SubtractVector(X, X\_q)) / VectorEnergyNorm(A, SubtractVector(X0, X\_q)) > eps1);

output << "w = " << w << " " << "Iter = " << iter << endl;

if (iter < iter\_best || iter\_best == -1) {

w\_best = w;

iter\_best = iter;

}

w += 0.1;

}

output << endl << "w\* = " << w\_best << " " << "IterMin = " << iter\_best << endl;

output << endl << "Метод ПВР" << endl;

output << setw(5) << "Iter |" << setw(10) << "w |" << setw(16) << "Норма невязки |"

<< setw(20) << "Норма погрешности |"

<< setw(9) << "X[1]" << setw(10) << "X[2]" << setw(10) << "X[3]" << setw(10) << "X[4]" << endl;

iter = 0;

for (int i = 0; i < n; i++)

X[i] = B[i];

do {

r = ResidualVector(A, X, B);

for (int i = 0; i < n; i++) {

double sum = 0;

for (int j = 0; j < i; j++)

sum = sum + A[i][j] \* X[j];

for (int j = i + 1; j < n; j++)

sum = sum + A[i][j] \* X[j];

X[i] = (1 - w\_best) \* X[i] + w\_best \* (B[i] - sum) / A[i][i];

}

norm\_error = VectorEnergyNorm(A, SubtractVector(X, X\_q));

norm\_resid = VectorEnergyNorm(A, r);

output << setw(3) << iter + 1 << " |" << setw(8) << w\_best << " |" << setprecision(7) << setw(12) << norm\_resid << " |" << setw(14) << norm\_error << " |";

for (int i = 0; i < n; i++)

output << setw(10) << fixed << setprecision(5) << X[i];

output << endl;

iter++;

} while (VectorEnergyNorm(A, SubtractVector(X, X\_q)) / VectorEnergyNorm(A, SubtractVector(X0, X\_q)) > eps);

output << endl << "Сравнение полученного решения с решением, полученным прямым методом" << endl << "x" << endl;

OutputVect(X);

output << endl << "x\*" << endl;

OutputVect(X\_q);

output << endl << "||x - x\*||" << endl;

OutputVect(SubtractVector(X, X\_q));

output << endl;

return iter;

}

int main()

{

double x;

double\*\* A = new double\* [n];

for (int i = 0; i < n; i++)

A[i] = new double[n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

input >> A[i][j];

double\* B = new double[n];

for (int i = 0; i < n; i++)

B[i] = i + 1;

double\* iter = new double[n];

for (int i = 0; i < n; i++)

iter[i] = 0;

output << "Variant = " << endl;

output << "b" << endl;

OutputVect(B);

output << "A" << endl;

MatrixOutput(A);

output << "Решение прямым методом квадратного корня" << endl << "x\*" << endl;

OutputVect(MethodSquareRoot(A, B));

output << endl << "Норма матрицы = " << MatrixNormCalc(A) << endl;

iter[0] = SimpleIterMethod(A, B);

iter[1] = SteepestDescentGradientMethod(A, B);

iter[2] = SOR(A, B);

iter[3] = ConjugateGradientMethod(A, B);

double cnc = CondNumberCalc(A);

output << endl << "Число обусловленности = " << setprecision(3) << cnc << endl;

output << endl << "Теоретическая оценка числа итераций" << endl;

output << "Методы простой итерации и градиентного спуска: " << fixed << setprecision(0) << log(1 / eps) / 2 \* cnc << endl;

output << "Метод релаксации: "<< fixed << setprecision(0) << log(1 / eps) / 4 \* sqrt(cnc) << endl;

output << "Метод сопряженных градиентов: " << fixed << setprecision(0) << log(2 / eps) / 2 \* sqrt(cnc) << endl;

output << endl << "Фактическое число итераций" << endl;

output << "Методы простой итерации: " << iter[0] << endl;

output << "Метод градиентного спуска: " << iter[1] << endl;

output << "Метод релаксации: " << iter[2] << endl;

output << "Метод сопряженных градиентов: " << iter[3] << endl;

output.close();

input.close();

}