МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу

«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 1

ОТЧЕТ

о выполненном задании

студента 205 учебной группы факультета ВМК МГУ

Ярёменко Григория Алексеевича

(фамилия, имя, отчество)

Подвариант 1: приложение 1-1, приложение 2 (п. 2-6)
Подвариант 2: приложение 1-1, приложение 2 (п. 2-6)

Москва, 2017 г.

Цель работы

Изучить классический метод Гаусса (а также модифицированный метод Гаусса), применяемый для решения системы линейных алгебраических уравнений.

Изучить классические итерационные методы (Зейделя и верхней релаксации), используемые для численного решения систем линейных алгебраических уравнений; изучить скорость сходимости этих методов в зависимости от выбора итерационного параметра.

Постановка задачи

Дана система уравнений $Ax=f$ порядка $n\times n$ с невырожденной матрицей A. Написать программу, решающую систему линейных алгебраических уравнений заданного пользователем размера (n – параметр программы) методом Гаусса и методом Гаусса с выбором главного элемента.

Предусмотреть возможность задания элементов матрицы системы и ее правой части как во входном файле данных, так и путем задания специальных формул.

Написать программу численного решения данной системы линейных алгебраических уравнений (n – параметр программы), использующую численный алгоритм итерационного метода Зейделя и верхней релаксации.

Цели и задачи практической работы

1) Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента;

2) Вычислить определитель матрицы det(A);

3) Вычислить обратную матрицу A^{-1};

4) Определить число обусловленности $M_A=||A||\times||A^{-1}||$;

5) Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра n);

6) Решить заданную СЛАУ итерационным методом Зейделя (или более общим методом верхней релаксации);

7) Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью;

8) Изучить скорость сходимости итераций к точному решению задачи (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра ω)

Алгоритм решения

Для реализации поставленных задач я задействовал 4 языковых среды:

- C 11: Front-end, оболочка для консольной утилиты
- Python 3: Сценарий, генерирующий матрицы
- Haskell 8: Back-end, матричные вычисления
- Bash: Front-end, главный сценарий

Для проверки корректности работы программы был задейстован интернет-ресурс https://www.wolframalpha.com/. Проверка корректности также реализована через применение функции невязки, в частности используемой в реализации иттерационных методов:

$$\mathbf{R(x) = \| Ax - b \|}$$

Достаточно убедиться в корректности работы функий нормы, умножения матрицы на вектор и векторного вычитания, чтобы при $R(x) < \alpha$, где α — погрешность вычислений в числах с плавающей точкой, утверждать, что вектор «x» соответствует решению СЛАУ (A | b).

Листинг вывода сценария sc1p1:

```
Solving with Gauss...
1:
Residual: 8.881784197001252e-16
Result: 6.000000e-01 1.000000e+00 -1.000000e+00 -2.000000e-01
2:
mtxprog: Prelude.last: empty list
```

```
3:
Residual: 0.0
Result: 3.000000e+00 2.000000e+00 1.000000e+00 0.000000e+00



Solving with Gauss LE...
1:
Residual: 0.0
Result: 6.000000e-01 1.000000e+00 -1.000000e+00 -2.000000e-01
2:
mtxprog: Prelude.last: empty list
3:
Residual: 0.0
Result: 3.000000e+00 2.000000e+00 1.000000e+00 0.000000e+00



Computing Gauss determinant...
1:
Result: 1.0000000000e+01
2:
Result: 0.0000000000e+00
3:
Result: -1.8900000000e+02



Computing invert matrix...
1:
Result:
-6.000000e-01 1.000000e-01 3.000000e-01 -2.000000e-01
1.000000e+00 5.000000e-01 -5.000000e-01 0.000000e+00
-1.000000e+00 1.500000e+00 -5.000000e-01 0.000000e+00
-8.000000e-01 3.000000e-01 -1.000000e-01 4.000000e-01
```

```
2:
mtxprog: Prelude.last: empty list
3:
Result:
-1.719577e+00 1.222222e+00 -6.560847e-01 8.624339e-01
-6.507937e-01 3.333333e-01 -2.698413e-01 5.079365e-01
-9.894180e-01 5.555556e-01 -4.021164e-01 5.608466e-01
-7.407407e-02 1.111111e-01 -1.851852e-01 7.407407e-02


Computing matrix condition number...
1:
Result: 5.7800000000e+01
2:
mtxprog: Prelude.last: empty list
3:
Result: 9.9582010582e+01
```

Сценарий sc1p1 выполняет все необходимые вычисления для достижений целей подварианта 1 относительно указанных данных приложения 1. Здесь СЛАУ 1, 2, 3 — это первая, вторая и третья матрицы варианта 1 приложения 1 соответственно. В случае вычисления числа обусловленности, определителя и обратной матрицы, вектор свободных членов игнорируется.

Как видно из листинга, программа успешно вычисляет требуемые величины. Программа заканчивает свою работу необработанным исключенем в случае некорректных данных. Числа представленны в научной нотации, чтобы сохранить принципиальную разницу между 0 и числами низких порядков (как например в случае вычисления определителя). Реализации методов Гаусса и Гаусса с выбором главного элемента предусматривают вывод значения невязки вычисленного вектора решений для исследования вопроса вычислительной устойчивости этих методов. Для вычисления обратной матрицы используется метод Жордана-Гаусса. В вычислении числа обусловленности используется 1-норма вектора.

Листинг вывода сценария sc1p2 1:

```
X = 1 | M = 6 | N = 100

Solving with Gauss...

Residual: 2.6506158752424127e-9

Result: -4.235932e-01 -4.095285e-01 -2.811349e-01 -2.051859e-01
-1.566060e-01 -1.229070e-01 -9.804281e-02 -7.881625e-02 -6.339523e-02
-5.065954e-02 -3.988753e-02 -3.059414e-02 -2.244177e-02 -1.518844e-02
-8.656418e-03 -2.712389e-03 2.745463e-03 7.796102e-03 1.250158e-02
1.691123e-02 2.106464e-02 2.499383e-02 2.872489e-02 3.227917e-02
3.567423e-02 3.892450e-02 4.204189e-02 4.503617e-02 4.791536e-02
5.068596e-02 5.335322e-02 5.592127e-02 5.839329e-02 6.077162e-02
6.305788e-02 6.525302e-02 6.735740e-02 6.937085e-02 7.129272e-02
7.312188e-02 7.485682e-02 7.649561e-02 7.803597e-02 7.947527e-02
8.081053e-02 8.203844e-02 8.315542e-02 8.415752e-02 8.504055e-02
8.579998e-02 8.643100e-02 8.692853e-02 8.728716e-02 8.750121e-02
8.756471e-02 8.747138e-02 8.721466e-02 8.678766e-02 8.618320e-02
8.539379e-02 8.441162e-02 8.322855e-02 8.183611e-02 8.022549e-02
7.838755e-02 7.631278e-02 7.399132e-02 7.141292e-02 6.856698e-02
6.544247e-02 6.202801e-02 5.831175e-02 5.428147e-02 4.992450e-02
4.522770e-02 4.017751e-02 3.475987e-02 2.896026e-02 2.276364e-02
1.615448e-02 9.116703e-03 1.633713e-03 -6.311659e-03 -1.473715e-02
-2.366107e-02 -3.310237e-02 -4.308059e-02 -5.361592e-02 -6.472923e-02
-7.644203e-02 -8.877657e-02 -1.017558e-01 -1.154034e-01 -1.297437e-01
-1.448022e-01 -1.606046e-01 -1.771781e-01 -1.945501e-01 -2.127494e-01
-2.318055e-01



Solving with Gauss LE...

Residual: 4.46188122404099e-14

Result: -4.235932e-01 -4.095285e-01 -2.811349e-01 -2.051859e-01
-1.566060e-01 -1.229070e-01 -9.804281e-02 -7.881625e-02 -6.339523e-02
-5.065954e-02 -3.988753e-02 -3.059414e-02 -2.244177e-02 -1.518844e-02
-8.656418e-03 -2.712389e-03 2.745463e-03 7.796102e-03 1.250158e-02
1.691123e-02 2.106464e-02 2.499383e-02 2.872489e-02 3.227917e-02
3.567423e-02 3.892450e-02 4.204189e-02 4.503617e-02 4.791536e-02
5.068596e-02 5.335322e-02 5.592127e-02 5.839329e-02 6.077162e-02
6.305788e-02 6.525302e-02 6.735740e-02 6.937085e-02 7.129272e-02
7.312188e-02 7.485682e-02 7.649561e-02 7.803597e-02 7.947527e-02
8.081053e-02 8.203844e-02 8.315542e-02 8.415752e-02 8.504055e-02
8.579998e-02 8.643100e-02 8.692853e-02 8.728716e-02 8.750121e-02
8.756471e-02 8.747138e-02 8.721466e-02 8.678766e-02 8.618320e-02
```

```
8.539379e-02 8.441162e-02 8.322855e-02 8.183611e-02 8.022549e-02
7.838755e-02 7.631278e-02 7.399132e-02 7.141292e-02 6.856698e-02
6.544247e-02 6.202801e-02 5.831175e-02 5.428147e-02 4.992450e-02
4.522770e-02 4.017751e-02 3.475987e-02 2.896026e-02 2.276364e-02
1.615448e-02 9.116703e-03 1.633713e-03 -6.311659e-03 -1.473715e-02
-2.366107e-02 -3.310237e-02 -4.308059e-02 -5.361592e-02 -6.472923e-02
-7.644203e-02 -8.877657e-02 -1.017558e-01 -1.154034e-01 -1.297437e-01
-1.448022e-01 -1.606046e-01 -1.771781e-01 -1.945501e-01 -2.127494e-01
-2.318055e-01




Computing Gauss determinant...

Result: 2.2391916196e-43




Computing invert matrix...

Written to mtx/inv.mtx




Computing matrix condition number...

Result: 1.1291503997e+04
```

Сценарий sc1p2 выполняет все необходимые вычисления для достижений целей подварианта 1 относительно указанных данных приложения 2. Здесь входная СЛАУ сгенерированна по шаблону, указанному в варианте 6 приложения 2 с параметром «x», переданным в качестве первого аргумента командной строки (в данном случае «1»).
Как видно из листинга, программа успешно вычисляет требуемые величины. Программа опускает вывод обратной матрицы (так как она 100x100), и вместо этого сохраняет ее в файл.

Листинг вывода сценария sc2p1 1.1 0.000001:

```
OMEGA = 1.1 | EPSILON = 0.000001

Solving with Successive Over-Relaxation...

1:

Diverges.

Result: -nan -nan -nan -nan

2:

Diverges.

Result: -nan -nan -nan -nan

3:

Computational failure.

Result: -nan -nan -nan -nan

Symmetric positive-definite SLE:

Iterations: 8

Result: 2.927981e-01 1.447149e-01 2.452957e-01 4.797142e-02
```

Сценарий sc2p1 выполняет все необходимые вычисления для достижений целей подварианта 2 относительно указанных данных приложения 1 с заданными ω и ε в качестве первого и второго аргументов командной строки соотвтетственно (в данном случае «1.1» и «0.000001»). Здесь СЛАУ 1, 2, 3 — это первая, вторая и третья матрицы варианта 1 приложения 1 соответственно. Входны данные также дополненны следующей симметрической, положительно определенной СЛАУ:

$$\begin{cases} 11x_1+1x_2+2x_3+3x_4=4 \\ 1x_1+31x_2+4x_3+5x_4=6 \\ 2x_1+4x_2+43x_3+6x_4=12 \\ 3x_1+5x_2+6x_3+61x_4=6 \end{cases}$$

Реализация метода верхней релаксации предусматривают вывод количества иттераций, потребовавшихся для вычисления вектора решений с заданной точностью. В случае если метод расходится, заместо количества иттераций выводится строка «Diverges.». В случае, если метод не применим к заданной СЛАУ (например если в ходе вычислений требуется взять обратную матрицу от вырожденной) , заместо количества иттераций выводится строка «Computational failure.». В частности «Computational failure.» выводится в случае, если в ε-окрестности решения нет ни одного числа с плавающей точкой (двойной точности).

Так как ни первая, ни вторая, ни третья матрицы соответствующих СЛАУ варианта 1 приложения 1 не являются положительно определенными (согласно https://www.wolframalpha.com/), вывод программы отражает, что попытки вычислить их векторы решений методом верхней релаксации успехом не увенчались.

Критерий остановки иттерационного процесса построен на основе вышупомянутой невязки (**R(x) = || Ax - b ||**). В случае, если процесс сходится, достаточно после каждой иттерации сравнивать невязку с указанной допустимой погрешностью ε, запуская следующую иттерацию лишь в том случае, если невязка все еще превышает требуемое значение. Таким образом, полученный ответ гарантированно имеет заданную точность.

Листинг вывода сценария sc2p2 1 1.1 0.000001:

```
X = 1 | M = 6 | N = 100 | OMEGA = 1.1 | EPSILON = 0.000001

Solving with Successive Over-Relaxation...

Computational failure.

Result: -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan -nan
-nan -nan -nan
```

Сценарий sc2p2 выполняет все необходимые вычисления для достижений целей подварианта 2 относительно указанных данных приложения 2 с заданными ω и ε в качестве второго и третьего аргументов командной строки

соотвтетственно (в данном случае «1.1» и «0.000001»). Здесь входная СЛАУ сгенерированна по шаблону, указанному в варианте 6 приложения 2 с параметром «x», переданным в качестве первого аргумента командной строки (в данном случае «1»).
Матрица сгенерированной СЛАУ вещественна и не симметрична ($A_{1n} - A_{n1} = 2(n - 1)\cdot 0.1$), следственно она не является эрмитовой, а значит и положительно опредленной. Что характерно, программа выводит строку «Computational failure.», подразумевая неприменимость заданного иттерационного процесса к данной СЛАУ.

Архитектура и устройство программы

Проект в несобранном виде состоит из 10 исходных файлов:

1. sc1p1 — главный сценарий (bash), соответствующий подварианту 1, приложению 1.
2. sc1p2 — главный сценарий (bash), соответствующий подварианту 1, приложению 2. Принимает 1 аргумент командной строки: параметр «x» для генерации вектора свободных членов.
3. sc2p1 — главный сценарий (bash), соответствующий подварианту 2, приложению 1. Принимает 2 аргумента командной строки: иттерационный параметр ω и допустимую погрешность ε соответственно.
4. sc2p2 — главный сценарий (bash), соответствующий подварианту 2, приложению 2. Принимает 3 аргумента командной строки: параметр «x» для генерации вектора свободных членов, иттерационный параметр ω и допустимую погрешность ε соответственно.
5. generate — сценарий (Python3) , генерирующий СЛАУ приложения 2, пример 2-6. Принимает 1 аргумент командной строки: параметр «x» для генерации вектора свободных членов.
6. Matrix.hs — back-end модуль (Haskell), содержащий средства работы с матричными вычислениями и построенные на основе этих средств вычислительные методы.
7. FFI.hs — back-end модуль (Hakell), содержащий средства построения межъязыковых интерфейсов.

8. CMatrix.hs — интерфейсный модуль (Haskell), сопоставляющий некоторым функциям модуля «Matrix.hs» прототипы языка C, предусматривая прямую и обратную конверсию типов двух языковых сред.
9. main.c — front-end модуль (C), реализующий I/O. Императивные языки лучше справляются с использованием интерфейса системных вызовов, т.к. этот интерфейс имеет императивную природу. Для этого и был задействован промежуточный модуль, написанный на языке C.
10. Makefile — данный makefile собирает исполняемый файл mtxprog из файлов исходного кода «Matrix.hs», «FFI.hs», «Cmatrix.hs» и «main.c». Для трансляции, сборки и компановки используется компилятор GHC (Glassgow Haskell Compiler 8.0.1).

Сам mtxprog принимает один параметр командной строки, определяющий режим:

```
USAGE: mtxprog <MODE>
Possible MODE values:
  0 - Solve by Gauss method
  1 - Solve by Gauss with leading element method
  2 - Compute determinant with Gauss method
  3 - Solve by Successive Over-Relaxation
  4 - Compute invert matrix
  5 - Compute condition number
```

После запуска в одном из допустимых режимов, программа построчно читает со стандартного потока ввода квадратную матрицу. В случае, если режим подразумевает работу со СЛАУ, вначале построчно считывается матрица системы, а затем вектор свободных членов. Размер матрицы определяется автоматически по первой ее строке.

СЛАУ варианта 1 приложения 1 находятся в подкаталоге mtx, в файлах «1.mtx», «2.mtx» и «3.mtx». Добавочная симметрическая, положительно определенная СЛАУ также находится в каталоге mtx, в файле «sp.mtx». Результат вычисления обратной матрицы в ходе выполнения сценария sc1p2 записывается в файл «inv.mtx» в том же каталоге. Все из перечисленных СЛАУ хранятся в формате, соответствующем корректным входным данным для mtxprog. В случае, если текущий режим работы mtxprog подразумевает работу лишь с матрицей (а подана СЛАУ), последняя строка (вектор свободных членов) игнорируется.

sc1p1:

```
#!/bin/bash
echo Solving with Gauss...
echo 1:
./mtxprog 0 < mtx/1.mtx
echo 2:
./mtxprog 0 < mtx/2.mtx
echo 3:
./mtxprog 0 < mtx/3.mtx
echo -e "\n"
echo Solving with Gauss LE...
echo 1:
./mtxprog 1 < mtx/1.mtx
echo 2:
./mtxprog 1 < mtx/2.mtx
echo 3:
./mtxprog 1 < mtx/3.mtx
echo -e "\n"
echo Computing Gauss determinant...
echo 1:
./mtxprog 2 < mtx/1.mtx
echo 2:
./mtxprog 2 < mtx/2.mtx
echo 3:
./mtxprog 2 < mtx/3.mtx
echo -e "\n"
echo Computing invert matrix...
echo 1:
./mtxprog 4 < mtx/1.mtx
echo 2:
./mtxprog 4 < mtx/2.mtx
echo 3:
./mtxprog 4 < mtx/3.mtx
echo -e "\n"
echo Computing matrix condition number...
echo 1:
./mtxprog 5 < mtx/1.mtx
echo 2:
./mtxprog 5 < mtx/2.mtx
echo 3:
./mtxprog 5 < mtx/3.mtx
```

sc1p2:

```
#!/bin/bash
echo -e "X = $1 | M = 6 | N = 100"
echo Solving with Gauss...
./generate 100 $1 | ./mtxprog 0
echo -e "\n"
echo Solving with Gauss LE...
./generate 100 $1 | ./mtxprog 1
echo -e "\n"
echo Computing Gauss determinant...
./generate 100 $1 | ./mtxprog 2
echo -e "\n"
echo Computing invert matrix...
./generate 100 $1 | ./mtxprog 4 > mtx/inv.mtx
echo Written to mtx/inv.mtx
echo -e "\n"
echo Computing matrix condition number...
./generate 100 $1 | ./mtxprog 5
echo -e "\n"
```

sc2p1:

```
#!/bin/bash
echo -e "OMEGA = $1 | EPSILON = $2"
echo Solving with Successive Over-Relaxation...
echo 1:
(cat mtx/1.mtx && echo $1 $2) | ./mtxprog 3
echo 2:
(cat mtx/2.mtx && echo $1 $2) | ./mtxprog 3
echo 3:
(cat mtx/3.mtx && echo $1 $2) | ./mtxprog 3
echo Symmetric positive-definite SLE:
(cat mtx/sp.mtx && echo $1 $2) | ./mtxprog 3
echo -e "\n"
echo -e "\n"
echo Computing Gauss determinant...
./generate 100 $1 | ./mtxprog 2
echo -e "\n"
echo Computing invert matrix...
./generate 100 $1 | ./mtxprog 4 > mtx/inv.mtx
echo Written to mtx/inv.mtx
echo -e "\n"
echo Computing matrix condition number...
./generate 100 $1 | ./mtxprog 5
echo -e "\n"
```

sc2p2:

```
#!/bin/bash
echo -e "X = $1 | M = 6 | N = 100 | OMEGA = $2 | EPSILON = $3"
echo Solving with Successive Over-Relaxation...
(./generate 100 $1 && echo $2 $3) | ./mtxprog 3
echo -e "\n"
```

generate:

```
#!/usr/bin/python3
from sys import argv
from math import exp, cos

M  = 6
Qm = 1.001 - (2 * M * 0.001)
n = int(argv[1])
x = float(argv[2])
f = lambda i, j, n: (Qm ** (i + j)) + (0.1 * (j - i)) if i != j else (Qm
- 1) ** (i + j)
g = lambda i, n, x: x * exp(x / i) * cos(x / i)

def buildMtx():
    return ([[f(i + 1, j + 1, n) for j in range(n)] for i in range(n)],
[g(i + 1, n, x) for i in range(n)])

A, f = buildMtx()
for i in A:
    print(" ".join(map(str, i)))
print(" ".join(map(str, f)))
```

Matrix.hs:

```
module Matrix where

import Data.List hiding (transpose)
import Debug.Trace

{-- Модуль с вычислительными методами --}

--Определитель Лапласа
detLaplace :: [[Integer]] -> Integer
detLaplace [[a]] = a
detLaplace a     = sum $ map (\x -> (head $ drop (x - 1) $ last a)…
* (detLaplace [take (x - 1) y ++ drop x y| y <- init a]) * ((-1)^ …
(x + n))) [1..n] where n = length a

--Все возможные перестановки
permute    :: [a] -> [[a]]
permute [a]      = [[a]]
permute (x:xs)   = concat $ map (\p -> [take p y ++ [x] ++ drop p …
y|y <- permute xs]) [0..n] where n = length xs

--Четность перестановки
parity     :: [Int] -> Int
parity [a]       = 1
parity (x:xs)    = parity xs * product (map (\p -> signum $ p - x) xs)

--Определитель пересчетом перестановок
detPerm    :: [[Int]] -> Int
detPerm   a      = sum $ map (\p -> parity p * (product $ zipWith …
(\c d -> head $ drop (c - 1) d) p a)) $ permute [1..n] where n =  …
length a

--Транспонирование
transpose  :: [[a]] -> [[a]]
trasnpose [[]]   = [[]]
transpose []     = []
transpose ([]:_) = []
transpose a      = (map head a):(transpose $ map tail a)

--Квантор всеобщности для списков
forAll     :: (a -> Bool) -> [a] -> Bool
```

```
forAll _ []       = True
forAll f (x:xs)   = f x && forAll f xs

--Квантор существования для списков
exists      :: (a -> Bool) -> [a] -> Bool
exists _ []       = False
exists f (x:xs)   = f x || exists f xs

--Определитель Гаусса
detGauss     :: (RealFloat a) => [[a]] -> a
detGauss [[a]]    = a
detGauss a@((x:_):_) | x == 0    = if exists (0/=) $ head $       …
transpose a then (*) (-1) $ detGauss $ head [(head $ drop y a):   …
(take y a ++ drop (y + 1) a)|y <- [1..], head (head $ drop y a)   …
 /= 0] else 0
                     | otherwise = (*) x $ detGauss $ map (\k ->  …
tail $ zipWith (\q p -> p - ((q * head k)/x)) (head a) k) $ tail a
--Перемножение матриц
mtxMult      :: (RealFloat a) => [[a]] -> [[a]] -> [[a]]
mtxMult a b       = map (\x -> map (\y -> sum $ zipWith (*) x y) $ …
transpose b) a

--Домножение матрицы на константу
mtxMultC     :: (RealFloat a) => a -> [[a]] -> [[a]]
mtxMultC a b      = map (map (a*)) b

--Сумма матриц
mtxAdd       :: (RealFloat a) => [[a]] -> [[a]] -> [[a]]
mtxAdd a b        = zipWith (zipWith (+)) a b

--Разность матриц
mtxSub       :: (RealFloat a) => [[a]] -> [[a]] -> [[a]]
mtxSub a b        = zipWith (zipWith (-)) a b

--Возведение матрицы в степень
mtxPow       :: (RealFloat a) => [[a]] -> Int -> [[a]]
mtxPow a 0        = mtxOne $ length a
mtxPow a n        = mtxMult a $ mtxPow a $ n - 1

--Евклидова норма вектора
vcrNorm2     :: (RealFloat a) => [a] -> a
vcrNorm2 a        = sqrt $ sum $ map (**2) a
```

```
--Евклидова норма матрицы
mtxNorm2      :: (RealFloat a) => [[a]] -> a
mtxNorm2 a        = vcrNorm2 $ concat a

--Нулевые матрицы
mtxZero       :: (Num a) => Int -> [[a]]
mtxZero n         = [[0|_<-k]|_<-k] where k = [1..n]

--Единичные матрицы
mtxOne        :: (RealFloat a) => Int -> [[a]]
mtxOne 1          = [[1]]
mtxOne n          = ([1] ++ [0|_<-[1..(n - 1)]]):(map ([0] ++)  …
$ mtxOne $ n - 1)

--Метод вычисления определителя по-умолчанию
det               = detGauss

--Преобразование вектора в квадратную матрицу
mtx           :: [a] -> [[a]]
mtx v             = [take n $ drop (n * x) v|x <- [0..(n - 1)]] …
where n = floor $ sqrt $ fromIntegral $ length v

--Отображение матрицы
mtxMap        :: (a -> b) -> [[a]] -> [[b]]
mtxMap f a        = map (map f) a

--Максимум по вектору
vcrMax        :: Ord a => [a] -> a
vcrMax a          = head [x| x <- a, forAll (x>=) a]

--Минимум по вектору
vcrMin        :: Ord a => [a] -> a
vcrMin a          = head [x| x <- a, forAll (x<=) a]

--Максимум по матрице
mtxMax        :: Ord a => [[a]] -> a
mtxMax a          = vcrMax $ concat a

--Минимум по матрице
mtxMin        :: Ord a => [[a]] -> a
```

```
mtxMin a         = vcrMin $ concat a

--Сумма набора матриц
mtxSum       :: (RealFloat a) =>  [[[a]]] -> [[a]]
mtxSum [a]       = a
mtxSum (x:xs)    = mtxAdd x $ mtxSum xs

--Произведение набора матриц
mtxProd      :: (RealFloat a) =>  [[[a]]] -> [[a]]
mtxProd [a]      = a
mtxProd (x:xs)   = mtxMult x $ mtxProd xs

--Многочлен из матриц
mtxPoly      :: (RealFloat a) =>  [a] -> [[a]] -> [[a]]
mtxPoly [c] a    = mtxMultC c $ mtxOne $ length a
mtxPoly c a      = mtxAdd (mtxPoly [head c] a) $ mtxMult a $       …
mtxPoly (tail c) a

--Союзная матрица
adj          :: [[Double]] -> [[Double]]
adj a            = [[let t q p = take (q - 1) p ++ drop q p in (*) …
((^) (-1) $ x + y) $ det $ t y $ map (t x) a| y <- [1..n]] | x <-  …
[1..n]] where n = length a

--Обращение через союзные матрицы
invAdj       :: [[Double]] -> [[Double]]
invAdj a           = mtxMap (/ det a) $ adj a

--Обращение Шульца
invSchultz   :: (RealFloat a) => Int -> a -> [[a]] -> [[a]]
invSchultz n err a = schultz n err a (mtxMultC ((/) 1 $        …
mtxNorm2 $ mtxMult a t) t) where t = transpose a
    schultz m err a u = let psi = mtxSub (mtxOne (length a)) $ …
mtxMult a u in if mtxNorm2 psi < err then u else schultz m err …
a (mtxMult u $ mtxPoly [1|_<-[0..m]] psi)

--Обращение Жордана-Гаусса
invGaussJordan :: (RealFloat a) => [[a]] -> [[a]]
invGaussJordan a0  = gauss $ jordan (a0, mtxOne $ length a0) where
    takeLast n xs = foldl (const . tail) xs (drop n xs)
    dropLast n xs = foldl (const . init) xs [1..n]
    gauss ([[1]], b) = b
```

```
    gauss (a, b)     = gauss (map init $ init a, (zipWith      …
(\x -> zipWith (\y z -> z - y*x) current) (init $ map last a)  …
$ take (length a - 1) b) ++ (drop (length a - 1) b)) where     …
current = last $ take (length a) b
    jordan  ([[a]], b) = ([[1]], init b ++ [map (/a) $ last b])
    jordan  (a@((x:_):_), b) | x == 0    = jordan $ head [let  …
swap m = (take (length m - length a) m) ++ [last $ dropLast    …
y m] ++ (drop (length m - length a) $ dropLast (y + 1) m ++    …
takeLast y m) in (swap a, swap b)|y <- [0..], head (last $     …
dropLast y a) /= 0]
                             | otherwise = ((head a'):(map (0:)…
$ fst next), snd next) where
        (a', b')   = ((map (/x) $ head a):(tail a), dropLast n …
b ++ [map (/x) $ head $ takeLast n b] ++ (takeLast (n - 1) b))
        (a'', b'') = (map (\k -> tail $ zipWith (\q p -> p — q …
* head k) (head a') k) $ tail a', dropLast (n - 1) b' ++       …
(zipWith (\k l -> zipWith (\q p -> p - q * head l) (head $     …
takeLast n b') k) (takeLast (n - 1) b') $ tail a'))
        n          = length a
        next       = jordan (a'', b'')

--Метод вычисления обратной матрицы по-умолчанию
inv                = invGaussJordan

--Умножение матрицы на вектор
opApply      :: (RealFloat a) => [[a]] -> [a] -> [a]
opApply o v      = head $ transpose $ mtxMult o $ transpose [v]

--Решение методом Крамера
cramer       :: [[Double]] -> [Double] -> [Double]
cramer a0 v      = [(/) (det (take (y - 1) a ++ [v] ++ drop y a)) …
$ det a|y <- [1..(length a)]] where a = transpose a0



--Маска матрицы
mtxMask      :: (Num a) => [[Bool]] -> [[a]] -> [[a]]
mtxMask m a = zipWith (zipWith (\x y -> if x then y else 0)) m a

--Служебная функция для упрощения рекурсивных методов
--Возвращает подматрицу данной матрицы, исключая первые столбец и строку
subMtx    :: [[a]] -> [[a]]
subMtx a = map tail $ tail a
```

```
--Служебная функция подставляющая подматрицу
mtxMerge     :: [[a]] -> [[a]] -> [[a]]
mtxMerge a b = head a : (zipWith (\x y -> head x : y) (tail a) b)

--Разложение матрицы на диагональную и треугольные
mtxDecomp    :: (Num a) => [[a]] -> ([[a]], [[a]], [[a]])
mtxDecomp [] = ([], [], [])
mtxDecomp a  = (d, l, u) where
    (sd, sl, su) = mtxDecomp $ subMtx a
    zm           = mtxZero $ length a
    d            = (((head (head a) : tail (head zm))) : tail zm) …
`mtxMerge` sd
    l            = (head zm : tail a) `mtxMerge` sl
    u            =  ((0 : tail (head a)) : tail zm) `mtxMerge` su

--Невязка
residual     :: [[Double]] -> [Double] -> [Double] -> Double
residual a x b = vcrNorm2 $ zipWith (-) (opApply a x) b

--Максимальная невязка
maxres = 1.0e50

--Верхняя релаксация
sccOvRl      :: Double -> Double -> [Double] -> [[Double]] -> …
[Double] -> (Int, [Double])
sccOvRl w eps x0 a b
    | res > maxres = (-1, map (const (0/0)) b)
    | res > eps    = let (i, x) = sccOvRl w eps current a b   …
in if i /= (-1) then (1 + i, x) else (i, x)
    | otherwise    = (0, current)
    where (d, l, u) = mtxDecomp a
          pr1       = inv $ d `mtxAdd` mtxMultC w l
          rhs       = opApply pr1 $ zipWith (-) b $ opApply a x0
          delta     = map (w*) rhs
          current   = zipWith (+) x0 delta
          res       = residual a current b

--Решение методом Гаусса
solveGauss  :: [[Double]] -> [Double] -> [Double]
solveGauss a f = reverse $ svTrg $ mkTrg a f where
```

```
    mkTrg [] _ = ([], [])
    mkTrg a  f = (mtxMerge newa nga, head newf : ngf) where
        af           = zipWith (\m v -> m ++ [v]) a f
        newaf        = af0 : (map newafn $ tail af) where
            af0       = head af
            newafn m  = zipWith (-) m $ map ((head m  / head af0)*) af0
        (newa, newf) = (map init newaf, map last newaf)
        (nga, ngf)   = mkTrg (subMtx newa) (tail newf)
    svTrg ([], _) = []
    svTrg (a,  f) = x : svTrg ((map init $ init a), newf) where
        x    = last f / (last $ last a)
        newf = zipWith (\m v -> v - (last m * x)) (init a) (init f)

--Применение перестановки (или их композиция)
permApply :: [Int] -> [a] -> [a]
permApply p s = map (\x -> s !! (x - 1)) p

--Обращение перестановки
permInvert :: [Int] -> [Int]
permInvert x = map snd $ sort $ zip x [1..(length x)]

--Транспозиция
permSwap :: Int -> Int -> Int -> [Int]
permSwap l n0 m0
    | n0 == m0  = [1..l]
    | otherwise = (take (n - 1) s) ++ [s !! (m - 1)] ++ (take (m - n - 1)
$ drop n s) ++ [s !! (n - 1)] ++ (drop m s) where
        (n, m) = (min n0 m0, max n0 m0)
        s      = [1..l]



--Решение методом Гаусса с выбором главного элемента
solveGaussLE  :: [[Double]] -> [Double] -> [Double]
solveGaussLE a f = permApply (permInvert prm2) $ reverse $ svTrg …
(na, nf) where
    maxPerms a = head [(permSwap l 1 x, permSwap l 1 y) | x <-   …
[1..l], y <- [1..l], abs (a !! (x - 1) !! (y - 1)) == mtxMax     …
(mtxMap abs a)] where
        l = length a
    mkTrg [] _   = ([], [], [], [])
    mkTrg ao fo  =  (rearTopRow ngp2 $ mtxMerge newa nga, head   …
newf : ngf, newp1, newp2) where
```

```
        rearTopRow p m = ((head $ head m) : (permApply p $ tail  …
$ head m)) : tail m
        (p1, p2)        = maxPerms ao
        f1              = permApply p1 fo
        a1              = transpose $ permApply p2 $ transpose $  …
permApply p1 ao
        af              = zipWith (\m v -> m ++ [v]) a1 f1
        newaf           = af0 : (map newafn $ tail af) where
            af0       = head af
            newafn m  = zipWith (-) m $ map ((head m  / head     …
af0)*) af0
        (newa, newf) = (map init newaf, map last newaf)
        (nga, ngf, ngp1, ngp2) = mkTrg (subMtx newa) (tail newf)
        (newp1, newp2)         = let comp x y = head x :         …
(permApply y $ tail x) in (comp p1 ngp1, comp p2 ngp2)
    svTrg ([], _) = []
    svTrg (a,  f) = x : svTrg ((map init $ init a), newf) where
        x    = last f / (last $ last a)
        newf = zipWith (\m v -> v - (last m * x)) (init a) (init f)
    (na, nf, prm1, prm2) = mkTrg a f

--1-норма вектора
vcrNorm1 :: [Double] -> Double
vcrNorm1 v = sum $ map abs v

--Норма, используемая при вычислении ЧО
mtxNormInf :: [[Double]] -> Double
mtxNormInf m = maximum $ map vcrNorm1 $ transpose m



--Число обусловленности
condNumber :: [[Double]] -> Double
condNumber a = (mtxNormInf a) * (mtxNormInf $ inv a)



--Красивый вывод матрицы
mtxDisp      :: (Show a) => [[a]] -> String
mtxDisp []            = ""
mtxDisp ([]:xs)       = "\n" ++ mtxDisp xs
mtxDisp ((x:xs):xxs) = (show x ++ " ") ++ mtxDisp (xs:xxs)
```

FFI.hs:

```
module FFI where
{-- Модуль со средствами постренния межъязыковых интерфейсов --}

import Foreign.C.Types
import Foreign.Marshal.Array
import Foreign.Marshal.Unsafe
import Foreign.Ptr
import Foreign.Storable
import Data.Typeable
import Unsafe.Coerce

--Преобразование типов. Используется для преобразования
--типов одного языка в типы второго.
coerce :: a -> b
coerce = unsafeCoerce

--Преобразовать массив заданной длинны в список
readVector :: Storable a => CInt -> Ptr a -> [b]
readVector len0 arr = map coerce rawlist where
                    len     = fromIntegral len0
                    rawlist = unsafeLocalState $ peekArray len arr

--Преобразовать двумерный массив заданных размеров в двумерный список
readMatrix :: Storable a => CInt -> CInt -> Ptr (Ptr a) -> [[b]]
readMatrix n0 m0 arr = map (readVector n0) ptrlist where
                    m       = fromIntegral m0
                    ptrlist = unsafeLocalState $ peekArray m arr

--Преобразовать вектор в массив
writeVector :: (Storable a, Storable b) => [a] -> Ptr b
writeVector v = unsafeLocalState $ newArray (map coerce v)

--Преобразовать двумерный список в двумерный массив
writeMatrix :: (Storable a, Storable b) => [[a]] -> Ptr (Ptr b)
writeMatrix m = unsafeLocalState $ newArray (map writeVector m)
```

CMatrix.hs:

```
{-# LANGUAGE ForeignFunctionInterface #-}

module CMatrix where
{-- Модуль с межъязыковым интерфейсом  --}

import FFI
import Matrix
import Foreign.C.Types
import Foreign.Ptr
import Debug.Trace

--C-прототип определителя по Гауссу
detGauss_hs :: Ptr (Ptr CDouble) -> CInt -> CDouble
detGauss_hs mtx0 n0 = coerce $ detGauss $ mtx where
                     mtx = readMatrix n0 n0 mtx0

--C-прототип решени СЛАУ по Гауссу + вывод невязки
solveGauss_hs :: Ptr (Ptr CDouble) -> Ptr CDouble -> CInt -> Ptr CDouble
solveGauss_hs mtx0 v0 n0 = writeVector $ (\x -> trace            …
("Residual: "  ++ show (residual mtx x v)) x) $ solveGauss mtx v where
    v   = readVector n0 v0
    mtx = readMatrix n0 n0 mtx0

--C-прототип решения СЛАУ по Гауссу с выбором г.э. + вывод невязки
solveGaussLE_hs :: Ptr (Ptr CDouble) -> Ptr CDouble -> CInt →    …
Ptr CDouble
solveGaussLE_hs mtx0 v0 n0 = writeVector $ (\x -> trace          …
("Residual: "  ++ show (residual mtx x v)) x) $  solveGaussLE mtx v where
    v   = readVector n0 v0
    mtx = readMatrix n0 n0 mtx0

--C-прототип решения СЛАУ верхней релаксацией + вывод кол-ва иттераций
sccOvRl_hs :: Ptr (Ptr CDouble) -> Ptr CDouble -> CInt -> Double …
-> Double -> Ptr CDouble
sccOvRl_hs mtx0 v0 n0 w eps = writeVector $ (\x -> trace (if fst …
x == (-1) then "Diverges." else if isNaN $ head $ snd x then     …
"Computational failure." else "Iterations: " ++ show (fst x)) $  …
snd x) $ sccOvRl (coerce w) (coerce eps) (map (const 0) v) mtx v …
where
    v   = readVector n0 v0
    mtx = readMatrix n0 n0 mtx0
```

```
--С-прототип обращения (Жордана-Гаусса)
inv_hs :: Ptr (Ptr CDouble) -> CInt -> Ptr (Ptr CDouble)
inv_hs mtx0 n0 = writeMatrix $ inv mtx where
    mtx = readMatrix n0 n0 mtx0

--С-прототип вычисления Ч.О.
condNumber_hs :: Ptr (Ptr CDouble) -> CInt -> CDouble
condNumber_hs mtx0 n0 = coerce $ condNumber mtx where
    mtx = readMatrix n0 n0 mtx0

--экспорт прототипов
foreign export ccall detGauss_hs :: Ptr (Ptr CDouble) ->     …
CInt -> CDouble
foreign export ccall solveGauss_hs :: Ptr (Ptr CDouble) ->   …
Ptr CDouble -> CInt -> Ptr CDouble
foreign export ccall solveGaussLE_hs :: Ptr (Ptr CDouble) -> …
Ptr CDouble -> CInt -> Ptr CDouble
foreign export ccall sccOvRl_hs :: Ptr (Ptr CDouble) ->      …
Ptr CDouble -> CInt -> Double -> Double -> Ptr CDouble
foreign export ccall inv_hs :: Ptr (Ptr CDouble) -> CInt ->  …
Ptr (Ptr CDouble)
foreign export ccall condNumber_hs :: Ptr (Ptr CDouble) ->   …
CInt -> CDouble
```

main.c:

```
#include <HsFFI.h> //заголовок для языковой интеграции
#ifdef __GLASGOW_HASKELL__
#include "CMatrix_stub.h" //автоматически сгенерированный заголовок
extern void __stginit_Cmatrix(void); //подкл. межъязыковой интерфейс
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
enum
{
      BUF_LEN = 30000,   //буффер для первой строки
      BASE = 100,        //изнач. Длинна массива для чисел одной строки
      EXP = 2,           //размер массива пр необхоимости удваивается
      MODE_SGAUSS   = 0, //режимы
      MODE_SGAUSSLE = 1,
      MODE_DGAUSS   = 2,
      MODE_SSOR     = 3,
      MODE_INV      = 4,
      MODE_CNNUM    = 5,
};

typedef struct { //квадратная матрица и ее размер
      double **mtx;
      int n;
} SqMtxWL;


SqMtxWL rdMtx(void) //считать матрицу
{
      char    buf[BUF_LEN];
      int     len  = BASE * sizeof(double);
      double *frow = malloc(len);
      int     i    = 0;
      int     res  = 0;
      int     cnt  = 0;
      fgets(buf, sizeof(buf), stdin); //размер опр. по первой строке
```

```
    while(sscanf(buf + cnt, "%lf%n", frow + i, &res) > 0) {
        cnt += res;
        i++;
        if(i == len) {
            len *= EXP;
            frow = realloc(frow, len);
        }
    }
    double **mtx = malloc(sizeof(double *) * i);
    mtx[0] = frow;
    for(int j = 1; j < i; j++) {
        mtx[j] = malloc(i * sizeof(double));
        for(int k = 0; k < i; k++) {
            scanf("%lf", &mtx[j][k]);
        }
    }
    SqMtxWL out = {mtx, i};
    return out;
}

double *rdVcr(int n) //считать вектор
{
    double *out = malloc(n * sizeof(double));
    for(int i = 0; i < n; i++) {
        scanf("%lf", out + i);
    }
    return out;
}

void printVector(double *v, int n) //вывести вектор
{
    for(int i = 0; i < n; i++) {
        printf("%.6e ", v[i]);
    }
    printf("\n");
}

void printMatrix(SqMtxWL m) //вывести матрицу
{
    for(int i = 0; i < m.n; i++) {
        printVector(m.mtx[i], m.n);
```

```
	}
}

int main(int argc, char *argv[])
{
	hs_init(&argc, &argv); //инициализировать Haskell back-end
#ifdef __GLASGOW_HASKELL__
	hs_add_root(__stginit_Cmatrix);
#endif
	const char usage[] = "USAGE: mtxprog <MODE>\nPossible MODE values:\n  0 - Solve by Gauss method\n  1 - Solve by Gauss with leading element method\n  2 - Compute determinant with Gauss method\n  3 - Solve by Successive Over-Relaxation\n  4 - Compute invert matrix\n  5 - Compute condition number\n"; //то, что выводится при некорректных аргументах к.с.
	if(argc == 1) {
		printf(usage);
		return 0;
	}
	SqMtxWL ipt;
	double *rhs;
	double *res;
	switch(atoi(argv[1])) { //выполнить задачи режима
		case MODE_SGAUSS: //метод Гаусса (СЛАУ)
			ipt = rdMtx();
			rhs = rdVcr(ipt.n);
			res = solveGauss_hs(ipt.mtx, rhs, ipt.n);
			printf("Result: ");
			printVector(res, ipt.n);
			break;
		case MODE_SGAUSSLE: //… с выбором главного элемента
			ipt = rdMtx();
			rhs = rdVcr(ipt.n);
			res = solveGaussLE_hs(ipt.mtx, rhs, ipt.n);
			printf("Result: ");
			printVector(res, ipt.n);
			break;
		case MODE_DGAUSS: //метод Гаусса (определитель)
			ipt = rdMtx();
			printf("Result: %.10e\n", detGauss_hs(ipt.mtx, ipt.n));
			break;
		case MODE_SSOR: //верхняя релаксация
			ipt = rdMtx();
```

```
                rhs = rdVcr(ipt.n);
                double omega, eps;
                scanf("%lf%lf", &omega, &eps);
                res = sccOvRl_hs(ipt.mtx, rhs, ipt.n, omega, eps);
                printf("Result: ");
                printVector(res, ipt.n);
                break;
            case MODE_INV: //обращение
                ipt = rdMtx();
                SqMtxWL temp = {inv_hs(ipt.mtx, ipt.n), ipt.n};
                printf("Result: \n");
                printMatrix(temp);
                break;
            case MODE_CNNUM: //число обусловленности
                ipt = rdMtx();
                printf("Result: %.10e\n", condNumber_hs(ipt.mtx,ipt.n));
                break;
            default: //неорректный идентификатор режима
                printf(usage);
                return 0;
        }
        hs_exit(); //выполнить завершающие процедуры Haskell среды.
        return 0;
}
```

Makefile:

```
CC = ghc
HSSDIRS =
HSSFLAGS =
all : mtxprog
mtxprog : main.c CMatrix.o Matrix.o FFI.o
	$(CC) --make -no-hs-main -optc-O $^ -o $@
CMatrix.o : CMatrix.hs Matrix.o FFI.o
	$(CC) $(HSSDIRS) $(HSSFLAGS) -c -O $<
Matrix.o : Matrix.hs
	$(CC) $(HSSDIRS) $(HSSFLAGS) -c -O $<
FFI.o : FFI.hs
	$(CC) $(HSSDIRS) $(HSSFLAGS) -c -O $<
clean :
	rm -f *.hi *.o CMatrix_stub.h mtxprog
```

Вывод

Из листинга вывода сценариев sc1p1 и sc1p2 можно сделать следующее нетривиальное наблюдение: невязка результата работы метода Гаусса достаточно резко (в худшую сторону) отличается от невязки результата работы его вариации. Исследуя возникшую на основе этого наблюдения гипотезу о более высокой вычислительной устойчивости метода Гаусса с выбором главного элемента, я провел серию тестов с матрицами больших размеров от 100x100. Результаты всех этих тестов были очень показательны: в каждом случае две невязки разнились в пять порядков (в пользу метода Гаусса с выбором главного элемента). Очевидно, это свидетельствует о том, что метод Гаусса с выбором главного элемента обладает гораздо большей вычислительной устойчивость, чем его классический аналог.

Серия тестов:

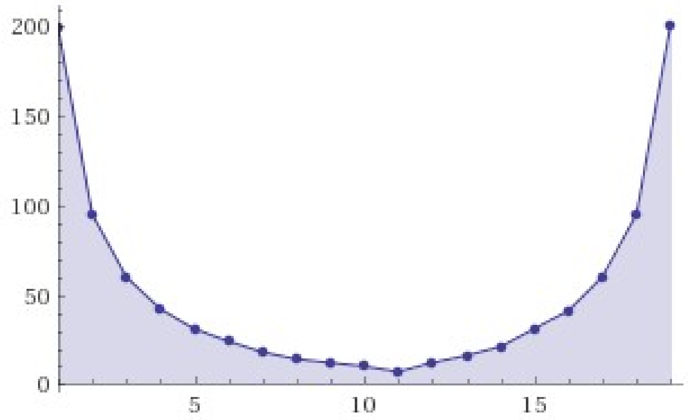
```
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 1 |
./mtxprog 0 | grep Residual
Residual: 2.6506158752424127e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 1 |
./mtxprog 1 | grep Residual
Residual: 4.46188122404099e-14
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 10 |
./mtxprog 1 | grep Residual
Residual: 4.206960004298459e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 10 |
./mtxprog 0 | grep Residual
Residual: 2.0340871113973765e-4
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 100
| ./mtxprog 0 | grep Residual
Residual: 5.836819688697503e36
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 100
| ./mtxprog 1 | grep Residual
Residual: 4.313778401356187e31
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 1 |
./mtxprog 0 | grep Residual
Residual: 2.6506158752424127e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 1 |
./mtxprog 1 | grep Residual
Residual: 4.46188122404099e-14
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 10 |
./mtxprog 0 | grep Residual
Residual: 2.0340871113973765e-4
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 10 |
./mtxprog 1 | grep Residual
```

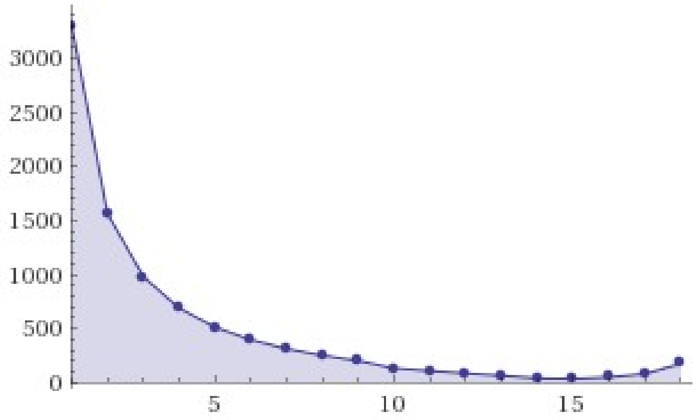
```
Residual: 4.206960004298459e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 100
| ./mtxprog 0 | grep Residual
Residual: 5.836819688697503e36
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 100 100
| ./mtxprog 1 | grep Residual
Residual: 4.313778401356187e31
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 1 |
./mtxprog 0 | grep Residual
Residual: 1.1260473293662082e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 1 |
./mtxprog 1 | grep Residual
Residual: 8.591605615587411e-14
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 10 |
./mtxprog 0 | grep Residual
Residual: 3.632335659303339e-5
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 10 |
./mtxprog 1 | grep Residual
Residual: 2.269217417570964e-9
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 100
| ./mtxprog 0 | grep Residual
Residual: 1.2442468046255202e36
grigory@debian:~/Documents/dev/general/sandbox_2$ ./generate 200 100
| ./mtxprog 1 | grep Residual
Residual: 3.0407479280476582e31
```

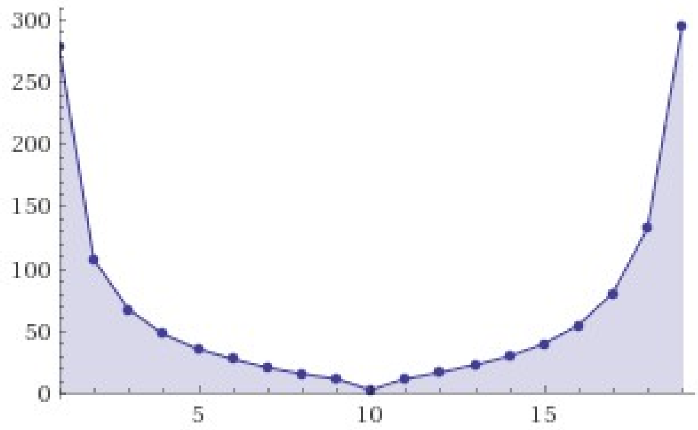
В ходе исследования скорости сходимости методов релаксации при разных значения ω я использовал 3 СЛАУ:

$$\begin{cases} 11x_1+1x_2+2x_3+3x_4=1 \\ 1x_1+31x_2+4x_3+5x_4=2 \\ 2x_1+4x_2+43x_3+6x_4=3 \\ 3x_1+5x_2+6x_3+61x_4=4 \end{cases} \qquad \begin{cases} 2x_1+1x_2=-1 \\ 1x_1+2x_2+1x_3=-2 \\ -x_2+2x_3=3 \end{cases} \qquad \begin{cases} 2x_1+6x_2=1 \\ 6x_1+20x_2=2 \end{cases}$$

Я применил к каждой из них релаксацию с ε = 0.00000001 и ω = 0.1·i, где i=1...19, с целью замерить кол-во затраченных иттераций и определить примерное расположение локальных минимумов. Были получены следующие результаты для трех систем соответственно:







Три системы имели следующие минимумы на данном наборе значений иттерационного параметра: 1.1, 1.6, 1. Из этого можно сделать вывод, что выбор иттерационного параметра индивидуален той СЛАУ, к которой он применяется, и «хорошего параметра» для произвольной системы не существует. Таки образом ответ на вопрос «дана СЛАУ с симметричной, положительно определенной матрицей; какой иттерационный параметр ω

имеет смысл взять для наиболее быстрой сходимости?» звучит следующим образом: «зависит от системы».