

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 206 учебной группы факультета ВМК МГУ
НИКОЛАЙЧУКА АРТЁМА КОНСТАНТИНОВИЧА

гор. Москва
2020 г.

Содержание

Цели	2
Постановка задачи	3
Описание алгоритмов	4
Описание и код программы	5
Тестирование	9
Тест 1	9
Тест 2	10
Тест 3	11
Тест 4	12
Тест 5	13
Тест 6	14
Выводы	15

Цели

Целью данной работы является реализация численных методов нахождения решения заданных систем линейных алгебраических уравнений методом Гаусса, в том числе методом Гаусса с выбором главного элемента, и методом верхней релаксации.

- Решить заданную СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента
- Вычислить определителю матрицы $\det(A)$
- Вычислить обратную матрицу A^{-1}
- Исследовать вопрос вычислительной устойчивости метода Гаусса
- Решить заданную СЛАУ итерационным методом верхней релаксации
- Разработать критерий остановки итерационного процесса для гарантированного получения приближенного решения исходной СЛАУ с заданной точностью
- Изучить скорость сходимости итераций к точному решению задачи при различных итреационных параметрах ω
- Проверить правильность решения СЛАУ на различных тестах, используя wolframalpha.com

Постановка задачи

Дана система линейных уравнений $A\bar{x} = \bar{f}$ порядка $n * n$ с невырожденной матрицей A .
Написать программу, решающую СЛАУ заданного пользователем размера методом Гаусса и методом Гаусса с выбором главного элемента.

Написать программу численного решения СЛАУ заданного пользователем размера, использующую итерационный метод верхней релаксации. Итерационный процесс имеет вид:

$$(D + \omega A^{(-)}) \frac{x^{k+1} - x^k}{\omega} + Ax^k = f),$$

где ω - итерационный параметр.

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

Описание алгоритмов

- Метод Гаусса

Цель - найти решение системы.

Решения системы линейных алгебраических уравнений методом Гаусса производится в два этапа: "прямой" и "обратный" ход. В результате "прямого хода" матрица коэффициентов, входящая в расширенную матрицу, путем линейных преобразований строк и их перестановок последовательно приводится к верхнему треугольному виду. На этапе "обратного хода" выполняется восстановление решения путем прохода по строкам матрицы в обратном направлении.

В методе Гаусса с выбором главного элемента на каждой итерации выбирается максимальный из всех элементов подматрицы, что позволяет уменьшить погрешность вычислений.

После приведения матрицы к диагональному виду с помощью классического метода Гаусса, определитель вычисляется, как произведение элементов, стоящих на диагонали.

Задача нахождения обратной матрицы решается с помощью метода Гаусса-Жордана. Над расширенной матрицей, составленной из столбцов исходной матрицы и единичной того же порядка, производятся преобразования метода Гаусса, в результате которых исходная матрица принимает вид единичной матрицы, а на месте единичной образуется матрица, обратная исходной.

- Метод верхней релаксации

Метод верхней релаксации это стационарным итерационным методом, в котором каждый следующий вектор приближения точного решения вычисляется по формуле:

$$(D + \omega T) \frac{y_{k+1} - y_k}{\omega} + Ay_k = f$$

где

ω - итерационный параметр,

y_k - k -й вектор приближения,

y_{k+1} - $(k + 1)$ -й вектор приближения,

A - матрица коэффициентов СЛАУ,

f - правая часть СЛАУ,

T - нижняя диагональная матрица матрицы A ,

D - матрица диагональных элементов матрицы A .

Метод применим только к положительноопределённым матрицам. По теореме Самарского для положительно определенных матриц A метод верхней релаксации сходится, если $0 < \omega < 2$. Значением по умолчанию является значение $\omega = \frac{4}{3}$. За вектор начального приближения берется нулевой вектор.

Описание и код программы

Весь код написан на языке программирования python.

Программа размещена на гитхабе. https://github.com/Strannik2001/Computing_Methods

Код основной программы:

```
1 import sys
2 import random
3 from math import sin
4
5
6 class Test:
7     def __init__(self, mode, n, a, b=1):
8         self.mode = mode
9         self.n = n
10        self.accuracy = 1e-9
11        self.max_iterations = 1000
12        self.omega = 4 / 3
13        self.need_optimization = b
14        self.matrix = []
15        self.b = []
16        self.q = 0
17        self.determinant = 1
18        self.invert_matrix = []
19        for i in range(n):
20            self.invert_matrix.append([])
21            self.invert_matrix[i] = [0] * self.n
22            self.invert_matrix[i][i] = 1
23        self.prepare_matrix(a)
24        self.matrix_copy = []
25        for i in range(n):
26            self.matrix_copy.append([])
27            for j in range(n):
28                self.matrix_copy[i].append(self.matrix[i][j])
29
30    def prepare_matrix(self, a):
31        if self.mode == 'static':
32            with open(a, "r") as f:
33                for i in range(self.n):
34                    self.matrix.append(list(map(float, f.readline().split())))
35                    self.b = list(map(float, f.readline().split()))
36        else:
37            for i in range(self.n):
38                self.matrix.append([])
39                self.matrix[i] = [0] * self.n
40            self.b = [0] * self.n
41            self.q = 1.001 - 2 * a / 1000
42            x = random.random()
43            for i in range(self.n):
44                self.b[i] = abs(x - self.n / 10) * (i + 1) * sin(x)
45            for i in range(1, self.n + 1):
46                for j in range(1, self.n + 1):
47                    if i == j:
48                        self.matrix[i - 1][j - 1] = pow(self.q - 1, i + j)
49                    else:
50                        self.matrix[i - 1][j - 1] = pow(self.q, i + j) + 0.1 * (j - i)
51
52    def out_matrix(self, a='matrix'):
53        if a == 'invert_matrix':
54            print("Invert_matrix")
55            m = self.invert_matrix
56        else:
57            print("Matrix")
58            m = self.matrix
```

```

59     out_matrix(m)
60
61 def get_invert_matrix(self):
62     if self.determinant != 0:
63         return self.invert_matrix
64     else:
65         print("No_such_matrix")
66         return None
67
68 def gaus(self):
69     sign = 1
70     matrix_copy = []
71     b_copy = []
72     for i in range(self.n):
73         b_copy.append(self.b[i])
74         matrix_copy.append([])
75         for j in self.matrix[i]:
76             matrix_copy[i].append(j)
77     for i in range(self.n):
78         if self.need_optimization:
79             maxi = i
80             for j in range(i, self.n):
81                 if abs(self.matrix[maxi][i]) < abs(self.matrix[j][i]):
82                     maxi = j
83             if maxi != i:
84                 sign *= -1
85                 self.b[i], self.b[maxi] = self.b[maxi], self.b[i]
86                 self.matrix[i], self.matrix[maxi] = self.matrix[maxi], self.matrix[i]
87                 self.invert_matrix[i], self.invert_matrix[maxi] = self.invert_matrix[maxi], self.
invert_matrix[i]
88             cur = self.matrix[i][i]
89             if cur == 0:
90                 self.determinant = 0
91                 continue
92             self.determinant *= cur
93             self.matrix[i] = [z / cur for z in self.matrix[i]]
94             self.b[i] /= cur
95             self.invert_matrix[i] = [z / cur for z in self.invert_matrix[i]]
96             for j in range(i + 1, self.n):
97                 cur = self.matrix[j][i]
98                 self.b[j] -= cur * self.b[i]
99                 self.matrix[j] = [self.matrix[j][z] - cur * self.matrix[i][z] for z in range(self.
n)]
100             self.invert_matrix[j] = [self.invert_matrix[j][z] - cur * self.invert_matrix[i][z]
for z in range(self.n)]
101     self.determinant *= sign
102     print("Determinant_{}".format(self.determinant))
103     for i in range(self.n)[::-1]:
104         for j in range(0, i):
105             self.invert_matrix[j] = [self.invert_matrix[j][z] - self.matrix[j][i] * self.
invert_matrix[i][z] for z in range(self.n)]
106             self.b[j] -= self.b[i] * self.matrix[j][i]
107             self.matrix[j] = [self.matrix[j][z] - self.matrix[j][i] * self.matrix[i][z] for z
in range(self.n)]
108     self.matrix = matrix_copy
109     if self.determinant == 0:
110         self.b = b_copy
111         print("No_invert_matrix((")
112         return
113     print("~~~~~")
114     print("Answer_by_gaus_method")
115     for j in range(self.n - 1):
116         print("%015.10f" % self.b[j], end=' | ^ | ')
117     print("%015.10f" % self.b[self.n - 1])
118     print("~~~~~")

```

```

119     self.b = b_copy
120
121 def relax_method(self):
122     res = [0] * self.n
123     t = [0] * self.n
124     for k in range(self.max_iterations):
125         for i in range(self.n):
126             s1 = 0
127             s2 = 0
128             for j in range(i):
129                 s1 += self.matrix[i][j] * t[j]
130             for j in range(i, self.n):
131                 s2 += self.matrix[i][j] * res[j]
132             t[i] = res[i] + (self.omega / self.matrix[i][i]) * (self.b[i] - s1 - s2)
133     d = 0
134     for i in range(self.n):
135         d += (res[i] - t[i]) * (res[i] - t[i])
136     if d < self.accuracy:
137         print("~~~~~")
138         print("Answer_by_relax_method")
139         for j in range(self.n - 1):
140             print("%015.10f" % t[j], end=' |^| ')
141         print("%015.10f" % t[self.n - 1])
142         print("~~~~~")
143         return t
144     for i in range(self.n):
145         res[i] = t[i]
146     print("~~~~~")
147     print("Answer_by_relax_method")
148     for j in range(self.n - 1):
149         print("%015.10f" % res[j], end=' |^| ')
150     print("%015.10f" % res[self.n - 1])
151     print("~~~~~")
152     return res
153
154
155 def out_matrix(a):
156     if a is None:
157         return
158     n = len(a)
159     if n > 20:
160         print('too_much_size')
161         return
162     print("#####")
163     for i in range(n):
164         for j in range(n - 1):
165             print("%015.10f" % a[i][j], end=' |^| ')
166         print("%015.10f" % a[i][n - 1])
167     print("#####")
168
169
170 def mul_matrix(a, b):
171     n = len(a)
172     res = []
173     for i in range(n):
174         res.append([])
175         for j in range(n):
176             res[i].append(0)
177             for k in range(n):
178                 res[i][j] += a[i][k] * b[k][j]
179     return res
180
181
182 def out_usage():
183     print("###Usage###")

```



```

184 print("First_argument_-'static'or'dynamic'")
185 print("Second_argument_-'matrix_order'")
186 print("Third_argument_-'path_to_file_if_static',_starting_number_if_dynamic")
187
188
189 def main():
190     try:
191         if sys.argv[1] == 'static':
192             matrix = Test('static', int(sys.argv[2]), sys.argv[3])
193         elif sys.argv[1] == 'dynamic':
194             matrix = Test('dynamic', int(sys.argv[2]), float(sys.argv[3]))
195         else:
196             out_usage()
197             exit(0)
198     except:
199         out_usage()
200         print("Type_parameters")
201         a = input().split()
202         if a[0] == 'static':
203             matrix = Test('static', int(a[1]), a[2])
204         elif a[0] == 'dynamic':
205             matrix = Test('dynamic', int(a[1]), float(a[2]))
206         else:
207             out_usage()
208             exit(0)
209     matrix.gaus()
210     matrix.relax_method()
211     out_matrix(matrix.get_invert_matrix())
212
213
214 if __name__ == "__main__":
215     main()

```

Тестирование

Тестирование проводится на наборах СЛАУ из приложения 1-11 и приложения 2-5 и тестах, которые я придумал сам.

Результат работы на невырожденных матрицах сравниваем с точным ответом, полученном на сайте wolframalpha.com.

Тест 1

$$\begin{cases} 4x_1 - 3x_2 + x_3 - 5x_4 = 7 \\ x_1 - 2x_2 - 2x_3 - 3x_4 = 3 \\ 3x_1 - x_2 + 2x_3 = -1 \\ 2x_1 + 3x_2 + 2x_3 - 8x_4 = -7 \end{cases}$$

Матрица является невырожденной и положительноопределённой.

```
Determinant = 134.99999999999997
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by gaus method
0002.0000000000 |^| 0001.0000000000 |^| -003.0000000000 |^| 0001.0000000000
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Answer by relax method
0001.9999699442 |^| 0000.9999625743 |^| -002.9999654430 |^| 0000.9999830833
Iterations - 27
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
Invert Matrix:
#####
0000.5333333333 |^| -000.0000000000 |^| -000.6000000000 |^| 0000.3333333333
0000.5185185185 |^| -000.2222222222 |^| -000.8888888889 |^| 0000.4074074074
-000.5407407407 |^| -000.1111111111 |^| 0000.9555555556 |^| -000.2962962963
0000.1925925926 |^| -000.1111111111 |^| -000.2444444444 |^| 0000.0370370370
#####
Cond num = 7.644444444444445
```

Число обусловленности равно 7.5. Определитель равен 135. Решение системы методом Гаусса с выбором главного элемента полностью совпадает с решением, которое выдаёт wolframalfa. Метод верхней релаксации достигает точности 10^{-5} быстрее всего за 27 итераций при параметре $\omega = \frac{2}{3}$.

Случай, когда матрица вырождена, программа также обрабатывает корректно:

Тест 2

$$\begin{cases} 2x_1 - x_2 + 3x_3 + 4x_4 = 5 \\ 4x_1 - 2x_2 + 5x_3 + 6x_4 = 7 \\ 6x_1 - 3x_2 + 7x_3 + 8x_4 = 9 \\ 8x_1 - 4x_2 + 9x_3 + 10x_4 = 11 \end{cases}$$

```
##Usage##
First argument - 'static' or 'dynamic'
Second argument - matrix order
Third argument - path_to_file if static, starting number if dynamic
Type parameters
static 4 StaticTests/test3.txt
Determinant = 0
No invert_matrix(((
No such matrix
```

Тест 3

$$\begin{cases} x_1 + 2x_2 + 4x_3 = 31 \\ 5x_1 + x_2 + 2x_3 = 29 \\ 3x_1 - 1x_2 + x_3 = 10 \end{cases}$$

Determinant = -27.0

^^

Answer by gaus method

0003.0000000000 |^| 0004.0000000000 |^| 0005.0000000000

^^

Invert Matrix:

#####

-000.1111111111 |^| 0000.2222222222 |^| 0000.0000000000

-000.0370370370 |^| 0000.4074074074 |^| -000.6666666667

0000.2962962963 |^| -000.2592592593 |^| 0000.3333333333

#####

Cond num = 3.3333333333333333

Число обусловленности равно 3.33333. Определитель равен -27 . Решение методом Гаусса и определитель совпадают с решением wolframalpha.

Тест 4

$$\begin{cases} 24x_1 + 6x_2 + -12x_3 = 16 \\ 6x_1 + 33x_2 + 6x_3 = 8 \\ -12x_1 + 6x_2 + 24x_3 = 4 \end{cases}$$

```
Determinant = 11664.0
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Answer by gaus method
```

```
0001.0370370370 |^| -000.0740740741 |^| 0000.7037037037
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Answer by relax method
```

```
0001.0370344384 |^| -000.0740829064 |^| 0000.7037016969
```

```
Iterations - 29
```

```
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
Invert Matrix:
```

```
#####
```

```
0000.0648148148 |^| -000.0185185185 |^| 0000.0370370370
```

```
-000.0185185185 |^| 0000.0370370370 |^| -000.0185185185
```

```
0000.0370370370 |^| -000.0185185185 |^| 0000.0648148148
```

```
#####
```

```
Cond num = 2.138888888888889
```

Число обусловленности равно 2.13. Определитель равен 11664. Метод верхней релаксации достигает точности 10^{-5} быстрее всего за 29 итераций при параметре $\omega = \frac{4}{3}$.

Тест 5

Матрица порядка $n = 50$ $a_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), i \neq j \\ (q_M - 1)^{i+j}, i = j \end{cases} \quad b_i = |x - \frac{n}{10}| \cdot i \cdot \sin x$

```
dynamic 50 0.5
Determinant = 5157.249999998595
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Answer by gaus method
-001.9172774456  |^|  -001.8545062586  |^|  -001.7917350716  |^|
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

too much size
Cond num = 5.7818798778419005

Process finished with exit code 0
```

Столбец решение полностью не влез. Он совпадает с полученным с помощью wolframalpha решением.
Начальные параметры $q_M = 0.5, x \in (0, 1)$
Число обусловленности равно 5.781. Определитель равен 5157.25.

Тест 6

$$\begin{cases} 5x_1 + 3x_2 = 1 \\ 2x_1 - x_2 = 0 \end{cases}$$

```
static 2 StaticTests/test8.txt
Determinant = -11.0
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Answer by gaus method
0000.0909090909  |^|  0000.1818181818
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Answer by relax method
0000.0909028029  |^|  0000.1818089119
Iterations - 8
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Invert Matrix:
#####
0000.0909090909  |^|  0000.2727272727
0000.1818181818  |^|  -000.4545454545
#####
Cond num = 2.2727272727272725
```

Число обусловленности равно 2.(27). Определитель равен -11 . Метод верхней релаксации достигает точности 10^{-5} быстрее всего за 11 итераций при параметре $\omega = \frac{2}{3}$.

Выводы

Метода Гаусса находит решение с высокой точностью, особенно, если использовать языки программирования с точной арифметикой.

Метод верхней релаксации даёт решение с высокой точностью уже всего через 10 – 20 итераций, что будет даёт значительную выгоду по сравнению с методами Гаусса. Однако множество применимости этого метода сильно уже, чем у методов Гаусса.