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

гор. Москва
2021 г.

Содержание

Цели **2**

Постановка задачи **3**

Описание алгоритмов **4**

Тестирование **5**

Выводы **11**

Код программы **12**

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 итерационных параметрах ω
- Проверить правильность решения СЛАУ на различных тестах

-

Постановка задачи

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

2. 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
  Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа.

  На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. Для этого среди элементов первого столбца матрицы выбирают ненулевой, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают, пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.
  На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

  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

  Система линейных уравнений

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + \ldots + a_{2n}x_n &= b_2 \\ & \ldots \\ a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{cases}$$

  приводится к виду $\begin{cases} P_{11}x_1 + P_{12}x_2 + \ldots + P_{1n}x_n + c_1 &= 0 \\ & \ldots \\ P_{n1}x_1 + P_{n2}x_2 + \ldots + P_{nn}x_n + c_n &= 0 \end{cases}$

  где $P_{ij} = -\frac{a_{ij}}{a_{ii}}$, $c_i = \frac{b_i}{a_{ii}}$. То есть все P_{ii} = -1.
  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.

Тестирование

Тестирование проводится на наборах СЛАУ из приложения 1-13 и приложения 2-4 и тестах, которые я придумал сам.
Результат работы на невырожденных матрицах сравниваем с точным ответом, полученном с помощью библиотеки numpy.
Заметим, что метод верхней релаксации не работает на большинстве тестов. В тесте 3 матрица вырожденная, а в Testps положительная симметричная. Также для теста с матрицей 100x100 я обрезал выходные данные, чтобы они поместились в отчет. Обратная матрица вычисляется каждый раз при подсчете числа обусловленности.

```
test1
matrix A=
[[ 3. -2.  2. -2.]
 [ 2. -1.  2.  0.]
 [ 2.  1.  4.  8.]
 [ 1.  3. -6.  2.]]
vector b=
[[ 8.]
 [ 4.]
 [-1.]
 [ 3.]]
my program solution gauss:  [ 2.  -3.  -1.5  0.5]
my program solution relax:  [ 3.83977340e+21  2.68090019e+22  1.05109737e+22
-1.00888633e+22] |
numpy solution:  [ 2.  -3.  -1.5  0.5]

my program det:  24.00000000000001 |
numpy det:  24.000000000000025

my program condition number:  229.99999999999997 |
numpy condition number: 229.99999999999986
```

```
test2
matrix A=
[[ 2.  3.  1.  2.]
 [ 4.  3.  1.  1.]
 [ 1. -7. -1. -2.]
 [ 2.  5.  1.  1.]]
vector b=
[[4.]
 [5.]
 [7.]
 [1.]]
my program solution gauss:  [-3. -5. 39. -7.] |
my program solution relax:  [ 1.74978037e+73  4.35085147e+73 -3.45240719e+74
7.80335335e+73] |
numpy solution:  [-3. -5. 39. -7.]

my program det:  1.9999999999999978 |
numpy det:  2.000000000000002

my program condition number:  528.0000000000011 |
numpy condition number: 527.9999999999995
```

```
test3
Singular matrix!
```

```
testps
matrix A=
[[8. 7. 7.]
 [7. 7. 5.]
 [7. 5. 9.]]
vector b=
[[4.]
 [4.]
 [9.]]
my program solution gauss:  [-8.6  6.1  4.3]
my program solution relax:  [-8.5999948   6.09999612  4.29999808] |
numpy solution:  [-8.6  6.1  4.3]

my program det:  10.0 |
numpy det:  9.999999999999993

my program condition number:  176.0 |
numpy condition number: 176.00000000000009
```

```
testp2
matrix A=
[[4.90000000e-05 1.07914666e+00 1.17229263e+00 1.26548658e+00
  1.35872818e+00 1.45201708e+00]
 [8.79146657e-01 2.40100000e-09 1.06548658e+00 1.15872818e+00
  1.25201708e+00 1.34535296e+00]
 [7.72292630e-01 8.65486582e-01 1.17649000e-13 1.05201708e+00
  1.14535296e+00 1.23873549e+00]
 [6.65486582e-01 7.58728176e-01 8.52017079e-01 5.76480100e-18
  1.03873549e+00 1.13216434e+00]
 [5.58728176e-01 6.52017079e-01 7.45352959e-01 8.38735488e-01
  2.82475249e-22 1.02563919e+00]
 [4.52017079e-01 5.45352959e-01 6.38735488e-01 7.32164340e-01
  8.25639190e-01 1.38412872e-26]]
vector b=
[[146.86939399]
 [ 89.08079043]
 [ 75.40526114]
 [ 69.37618934]
 [ 65.99267266]
 [ 63.82917551]
 [ 62.32738267]
 [ 61.22427221]
 [ 60.37981297]
 [ 59.71263954]]

my program solution gauss:  [-86.77332727 -28.6338834  -14.84893734  -8.75717686 -5.32866692
  -3.12923449  -1.59711479  -0.46765487   0.40003438   1.08780167] |
my program solution relax:  [ 9.99111524e+005 -1.21932039e+014  2.98998271e+026 -1.47302936e+
1.45792360e+064 -2.89886059e+089  1.15791706e+119 -9.29123172e+152
  1.49762399e+191 -4.84901663e+233] |
numpy solution:  [-86.77332727 -28.6338834 -14.84893734  -8.75717686  -5.32866692
 -3.12923449  -1.59711479  -0.46765487   0.40003438   1.08780167]

my program det:  5.474482228062905e-26 |
numpy det:  5.474482228062891e-26

my program condition number:  5504.431157246059 |
numpy condition number: 5504.43115823677
```

```
tmp = [i for i in np.arange(0, 2, 0.01)]
A, b = read_data('testps')
X = Solve(A, b)
counts = []
for x in tmp:
    counts.append((X.relax(x, 0.001, 20000)[1], x))
print('best omega is ', min(counts)[1], 'steps=', min(counts)[0])
best omega is  1.58 steps= 30
```

Выводы

Метода Гаусса находит решение с высокой точностью.

Метод верхней релаксации позволяет находить решение с заданной точностью, однако множество применимости этого метода сильно уже, чем у методов Гаусса.

Код программы

```
import numpy as np
def read_data(name): # Загрузка матрицы из файла
  tmp = np.loadtxt(name)
  A = tmp[:, :-1]
  b = tmp[:, -1]
  return A, b

def make_matrix(n, M = 4, x = 1): # Создание матрицы по формуле из условия
  q = 1.001 - 2 * M * 0.001
  A = np.zeros((n, n))
  for i in np.arange(n):
    for j in np.arange(n):
      if i != j:
        A[i, j] = q**(i + j + 2) + 0.1 * (j - i)
      else:
        A[i, j] = (q - 1) ** (i + j + 2)
  b = np.zeros(n)
  for i in np.arange(n):
    b[i] = n * np.exp(x / (i+1)) * np.cos(x)
  return A, b

class Solve: # класс решения СЛАУ
  def __init__(self, A, b, y=0):
    self.A = A
    self.b = b
    self.inv = None
    self.n = b.shape[0]
    self.C = self.A.copy()
    self.y = self.b.copy()
    self.x = np.zeros(self.n)
    self.transposition = list(np.arange(self.n))
    self.det = 1

  def gauss(self): # классический метод Гаусса
    for i in np.arange(self.n):
      for x in np.arange(i, self.n):
        if self.C[x, i] != 0:
          self.C[[x,i]] = self.C[[i,x]]
          break
      x = self.C[i, i]
      if x == 0:
        print('Singular matrix')
        return -1
      self.det *= x
      self.C[i] /= x
      self.y[i] /= x

      for j in np.arange(i+1, self.n):
        first = self.C[j, i]
        self.C[j] -= self.C[i] * first
        self.y[j] -= self.y[i] * first

  def gauss_m(self): # Метод Гаусса с выбором максимального элемента
    for i in np.arange(self.n):
      piv = np.argmax(np.abs(self.C[i]))
      ind_i = self.transposition.index(i)
      ind_piv = self.transposition.index(piv)
      self.transposition[ind_i] = piv
      self.transposition[ind_piv] = i
      self.C[:,[i, piv]] = self.C[:,[piv, i]]
      if ind_i != ind_piv:
        self.det *= -1
```

```
63          x = self.C[i, i]
64          if x == 0:
65            self.det = 0
66            print('Singular matrix')
67            return -1
68          self.det *= x
69          self.C[i] /= x
70          self.y[i] /= x
71
72          for j in np.arange(i+1, self.n):
73            first = self.C[j, i]
74            self.C[j] -= self.C[i] * first
75            self.y[j] -= self.y[i] * first
76
77      def gauss_reverse(self): # обратный ход
78        for i in np.arange(self.n - 1, -1, -1):
79          self.x[i] = self.y[i]
80          for j in np.arange(i+1, self.n):
81            self.x[i] -= self.x[j] * self.C[i, j]
82
83      def compute_det(self): # подсчет определителя
84        if ((self.C == self.A).all):
85          self.gauss_m()
86        return self.det
87
88      def solve(self, mode): # функция для вызова решения
89        if (mode == 0):
90          self.gauss()
91        else:
92          self.gauss_m()
93        self.gauss_reverse()
94        return np.array([self.x[self.transposition[i]] for i in np.arange(self.n)])
95
96      def inverse(self): # нахождение обратной матрицы
97        self.inv = np.eye(self.n, dtype = float)
98        tmp = self.A.copy()
99        for i in np.arange(self.n):
100         for x in np.arange(i, self.n):
101           if tmp[x, i] != 0:
102             tmp[[x,i]] = tmp[[i,x]]
103             self.inv[[x,i]] = self.inv[[i,x]]
104             break
105         x = tmp[i, i]
106         tmp[i] /= x
107         self.inv[i] /= x
108
109         for j in np.arange(i+1, self.n):
110           first = tmp[j][i]
111           tmp[j] -= tmp[i]*first
112           self.inv[j] -= self.inv[i]*first
113       for i in np.arange(self.n - 1, -1, -1):
114         x = tmp[i, i]
115         for j in np.arange(i - 1, -1, -1):
116           first = tmp[j, i]
117           tmp[j] -= tmp[i] * first
118           self.inv[j] -= self.inv[i] * first
119       return self.inv
120
121     def compute_norm(self, A): # подсчет нормы матрицы бесконечная()
122       return np.linalg.norm(A, np.inf)
123
124     def condition_number(self): # подсчет числа обусловленности
125       if (self.inv == None):
126         self.inverse()
```

```
127     return self.compute_norm(self.A) * self.compute_norm(self.inv)
128
129
130   def relax(self, omega = 4/3, epsilon = 0.000001, max_iter = 2000): # метод верхней
        релаксации
131     cur = np.zeros(self.n, dtype=float)
132     count = 0
133     while self.compute_norm(self.A @ cur - self.b) > epsilon and count < max_iter:
134       for i in np.arange(self.n):
135         sum1 = 0
136         for j in np.arange(self.n):
137           if i != j:
138             sum1 += self.A[i, j] * cur[j]
139         cur[i] = cur[i] * (1 - omega) + (self.b[i] - sum1) * omega / self.A[i, i]
140       count += 1
141     return cur, count
```