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

Фазылова Рамазана Рамилевича

гор. Москва

2020 год

Содержание

Решение СЛАУ методом Гаусса

Постановка задачи

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

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

Описание алгоритмов решения

Метод Гаусса

Прямой ход:

1. Рассмотрим первую строку матрицы коэффициентов A.
2. Разделим все элементы этой строки и f_1 на ненулевой элемент a_{11}, при этом, если $a_{11} = 0$, найдем строку с ненулевым первым элементом(а она найдется, в силу невырожденности матрицы A), и поменяем эти строки местами.
3. Вычтем из последующих $n - 1$ строк первую строку, домноженную на первый элемент соответствующей строки
4. Таким образом, мы занулили все нижнедиагональные элементы первого столбца матрицы A
5. Повторим данную процедуру для оставшихся $n - 1$ строк
6. В итоге, получим нижнетреугольную матрицу коэффициентов C с единицами на главной диагонали и вектор-столбец $\widetilde{f}$, которые образуют СЛАУ $Cx = \widetilde{f}$, эквивалентную $Ax = f$

Обратный ход:

1. Последовательно вычисляем неизвестные x_i, начиная с $i = n$:
   $x_n = \widetilde{f}_n$
   $x_{n-1} = \widetilde{f}_{n-1} - c_{n-1,n} * x_n$
   ...
   $x_1 = \widetilde{f}_1 - \sum_{k=2}^{n} c_{1,k} * x_k$

Модифицированный метод Гаусса

В прямом ходе обычного метода Гаусса разделим на ведущий элемент - максимальный по модулю в строке. Главный элемент с помощью перестановки столбцов матрицы коэффициентов поставим на i-ое место. Таким образом, все элементы матрицы C по модулю будут меньше единицы, вследствие чего увеличится вычислительнаю точность. При этом, в обратном ходе необходимо будет учесть, что при перестановке столбцов A соответствующим образом меняется порядок неизвестных.

Вычисление определителя матрицы

Приведем матрицу к верхнетреугольному виду по прямому ходу метода Гаусса. При этом заметим, что

$$\det A = \det C * (-1)^k * p_1 ... p_n = (-1)^k * p_1 ... p_n,$$

где k - число перестановок строк/столбцов в методе Гаусса,
p_i - элементы, на которые делили строки в методе Гаусса.

Вычисление обратной матрицы методом Гаусса-Жордана

Присоединим к матрице коэффициентов единичную матрицу и применим прямой ход метода Гаусса к полученной матрице. Затем, аналогично прямому ходу, занулим элементы выше диагонали. В итоге получим слеудющее преобразование:

$$\left[\begin{array}{c|c} & 1 \\ A & \ddots \\ & 1 \end{array}\right] \xrightarrow{G.-J.} \left[\begin{array}{c|c} 1 & \\ \ddots & A^{-1} \\ 1 & \end{array}\right]$$

Описание программы

Некоторые вспомогательные функции

```
from math import exp, cos, sqrt
from copy import copy, deepcopy

def argmax(arr): # Первый максимальный по модулю элемент
    abs_arr = [abs(x) for x in arr]
    return abs_arr.index(max(abs_arr))

def mat_norm(matrix): # вычисление матричной нормы (бесконечная норма)
    return max(map(lambda x: sum(map(abs, x)), matrix))

def generate_slau(M, n, x): # СЛАУ по формуле из приложения 2 п.2 в. 4
    A = [[0]*n for x in range(n)]
    b = [0]*n
    q = 1.001 - 2*M*pow(10,-3)
    for i in range(1, n+1):
        for j in range(1, n+1):
            if i != j:
                A[i-1][j-1] = pow(q, i+j)+0.1*(j-i)
            else:
                A[i-1][j-1] = pow(q-1, i+j)
        b[i-1] = n*exp(x/i)*cos(x)
    S = Slau(n)
    S.A = A
    S.b = b
    return S

# вычисление верхней границы погрешности в методе верхней релаксации
def error(sl, x):
    A = sl.A
    f = sl.b
    n = sl.n
    err = [0]*n
    for i in range(n):
        for j in range(n):
            err[i] += A[i][j]*x[j]
        err[i] -= f[i]
    return norm(err)*mat_norm(sl.inv)

def norm(x): # евклидова норма вектора
    return sqrt(sum(map(lambda x: x**2, x)))
```

Описание класса СЛАУ

```
class Slau: # класс СЛАУ
    def __init__(self, a, y=0): # Инициализация СЛАУ
        if type(a) == str: # Через текстовый файл
            filename = a
            with open(filename, 'r') as f:
                n = int(f.readline())
                A = [[int(x) for x in f.readline().split()] for r in range(n)]
                b = [int(x) for x in f.read().split()]
                self.n = n
            self.A = A
            self.b = b
            self.n = len(A)
        elif type(a) == Slau: # Копия существующей СЛАУ
            matrix = a
            self.A = matrix.copy_A()
            self.b = matrix.copy_b()
            self.n = matrix.n
        elif type(a) == int: # Нулевая СЛАУ
            n = a
            self.A = [[0]*n for row in range(n)]
            self.b = [0]*n
            self.n = n
        elif type(a) == list: # Через матрицу A и вектор столбец y
            self.A = deepcopy(a)
            self.b = copy(y)
            self.n = len(a)
        self.init_solution()


    def init_solution(self):
        self.x = [0]*self.n # решение СЛАУ
        self.transposition = list(range(self.n)) # перестановки столбцов
        self.k = 0 # количество перестановок
        self.C = self.copy_A() # треугольная матрциа после метода Гаусса
        self.y = self.copy_b() # правая часть после метода Гаусса
        self.det = 1 # инициализация определителя

    def copy_A(self): # Копия матрицы коэффициентов
        return [list(a) for a in self.A]

    def copy_b(self): # Копия вектора y
        return list(self.b)

    def swap_col(self, a, b, matrix): # Поменять столбцы местами
```

```
45          if matrix == self.C: # учёт перестановки решений
46              if a != b:
47                  self.k+=1
48              ind_a = self.transposition.index(a)
49              ind_b = self.transposition.index(b)
50              self.transposition[ind_a] = b
51              self.transposition[ind_b] = a
52          for i in range(self.n):
53              tmp = matrix[i][a]
54              matrix[i][a] = matrix[i][b]
55              matrix[i][b] = tmp
56
57      def gauss_straight(self, modified = 0): # Прямой ход метода Гаусса
58          for i in range(self.n):
59              if modified: # Модифицированный метод Гаусса
60                  ind = argmax(self.C[i]) # поиск максимального элемента в строке
61                  self.swap_col(i, ind, self.C) # перестановка столбцов
62              else: # Обычный метод Гаусса
63                  for x in range(i+1, self.n): # поиск строки с ненулевым iым элементом и перестано
64                      if self.C[x][i] != 0:
65                          tmp = self.C[i]
66                          self.C[i] = self.C[x]
67                          self.C[x] = tmp
68                          tmp = self.y[i]
69                          self.y[i] = self.y[x]
70                          self.y[x] = tmp
71                          if i != x:
72                              self.k+=1
73                          break
74              el = self.C[i][i]
75              if el == 0:
76                  print('Вырожденная матрица')
77                  return -1
78
79              for j in range(self.n): # деление всех элементов строки на C[i][i]
80                  self.C[i][j] /= el
81
82              self.y[i] = self.y[i] / el
83              self.det *= el # подсчёт определителя
84
85              for j in range(i+1, self.n): # вычитание i-ой строки из последующих строк
86                  first = self.C[j][i]
87                  for k in range(i, self.n):
88                      self.C[j][k] -= self.C[i][k]*first
89                  self.y[j] -= self.y[i]*first
90
```

```
91      def gauss_reverse(self): # Обратный ход метода Гаусса
92          for i in reversed(range(self.n)):
93              self.x[i] = self.y[i]
94              for j in range(i+1, self.n):
95                  self.x[i] -= self.x[j]*self.C[i][j]
96
97      def solve(self, modified = 0): # Решить СЛАУ
98          self.init_solution()
99          self.gauss_straight(modified)
100         self.gauss_reverse()
101         self.det_c() # досчитать определитель после метода Гаусса
102         old_x = list(self.x)
103         for i in range(self.n): # учесть перестановку неизвестных
104             self.x[i] = old_x[self.transposition[i]]
105
106         return self.x
107
108     def det_c(self): # досчитать определитель
109         self.det *= pow(-1, self.k)
110
111     # Вычисление обратной матрицы
112     def inverse(self):
113         self.inv = [[0]*self.n for row in range(self.n)] # инициализация обратной матрицы
114         for i in range(self.n):
115             self.inv[i][i] = 1
116
117         tmp = self.copy_A()
118         # Приведение левой части расширенной матрицы к верхнетреугольному виду по методу Гаусса
119         for i in range(self.n):
120             # поиск строки с ненулевым iым элементом и перестановка строк
121             for x in range(i+1, self.n):
122                 if tmp[x][i] != 0: # перестановка строк местами
123                     cpy = tmp[i]
124                     tmp[i] = tmp[x]
125                     tmp[x] = cpy
126                     cpy = self.inv[i]
127                     self.inv[i] = self.inv[x]
128                     self.inv[x] = cpy
129                     break
130             el = tmp[i][i]
131
132             for j in range(self.n):
133                 tmp[i][j] /= el
134                 self.inv[i][j] /= el
135
136             for j in range(i+1, self.n):
```

```
                first = tmp[j][i]
                for k in range(self.n):
                    tmp[j][k] -= tmp[i][k]*first
                    self.inv[j][k] -= self.inv[i][k]*first
        for i in reversed(range(self.n)): # Приведение левой части матрицы к диагональному виду
            el = tmp[i][i]

            for j in reversed(range(0, i)):
                first = tmp[j][i]
                for k in range(0, self.n):
                    tmp[j][k] -= tmp[i][k]*first
                    self.inv[j][k] -= self.inv[i][k]*first
        return self.inv

    def condition_number(self): # число обусловленности
        self.inverse()
        return mat_norm(self.A)*mat_norm(self.inv)

    def print_stats(self, solve_method = 0): # вывести базовую информацию по СЛАУ
        self.solve(solve_method)
        print('Определитель: ', self.det)
        print('Число обусловленности: ', self.condition_number())
        print('Корни: ', self.x)
```

Тестирование программы

Для тестирования использовались СЛАУ из варианта 11 приложения 1, а так же СЛАУ с матрицей, заданной по специальной формуле из примера 2 приложения 2.

1. Тест 11-1:

$$A|b = \left[\begin{array}{cccc|c} 4 & -3 & 1 & 5 & 7 \\ 1 & -2 & -2 & -3 & 3 \\ 3 & -1 & 2 & 0 & -1 \\ 2 & 3 & 2 & -8 & -7 \end{array}\right]$$

Результат работы программы:
Определитель: 135.0000
Число обусловленности: 30.5556
Корни: 2.0000, 1.0000, -3.0000, 1.0000
Обратная матрица:

$$A^{-1} = \begin{bmatrix} 0.5333 & 0.0000 & -0.6000 & 0.3333 \\ 0.5185 & -0.2222 & -0.8889 & 0.4074 \\ -0.5407 & -0.1111 & 0.9556 & -0.2963 \\ 0.1926 & -0.1111 & -0.2444 & 0.0370 \end{bmatrix}$$

Проверка с помощью WolframAlpha:
Корни: 2, 1, -3, 1

2. Тест 11-2:
Матрица в примере 11-2 вырожденная, поэтому поменял в ней элементы a_{13} и a_{14}

$$A|b = \left[\begin{array}{cccc|c} 2 & -2 & -1 & 1 & 1 \\ 1 & 2 & -1 & 1 & 1 \\ 4 & -10 & 5 & -5 & 1 \\ 2 & -14 & 7 & -11 & -1 \end{array}\right]$$

Результат работы программы:
Определитель: -144.0000
Число обусловленности: 45.3333
Корни: 0.6666, 0.1666, 0, 0
Обратная матрица:

$$A^{-1} = \begin{bmatrix} 0.0000 & 0.5556 & 0.1111 & 0.0000 \\ -0.2500 & 0.3889 & 0.0278 & 0.0000 \\ -0.5000 & -0.1667 & 0.4167 & -0.2500 \\ -0.0000 & -0.5000 & 0.2500 & -0.2500 \end{bmatrix}$$

Проверка с помощью WolframAlpha:
Корни: 0.66666, 0.1666, -0.0000, -0.0000

3. Тест 11-3:
   Матрица в примере 11-3 вырожденная, поэтому в первой строке переместил a_{11} в конец, остальные элементы строки сдвинулись влево. Аналогично сделал для 3ей строки

$$A|b = \left[\begin{array}{cccc|c} -1 & 3 & 4 & 2 & 5 \\ 4 & -2 & 5 & 6 & 7 \\ -3 & 7 & 8 & 6 & 9 \\ 8 & -4 & 9 & 10 & 11 \end{array}\right]$$

   Результат работы программы:
   Определитель: -24.0000
   Число обусловленности: 310.0000
   Корни: -4.9999, -4.9999, 3.9999, -0.5
   Обратная матрица:

$$A^{-1} = \left[\begin{array}{cccc} -2.0000 & -4.5000 & 1.0000 & 2.5000 \\ -2.1667 & -4.3333 & 1.1667 & 2.3333 \\ 1.8333 & 2.1667 & -0.8333 & -1.1667 \\ -0.9167 & -0.0833 & 0.4167 & 0.0833 \end{array}\right]$$

   Проверка с помощью WolframAlpha:
   Корни: -5, -5, 4, -0.5

4. Тест 4:
   В данном тесте проверялась работоспособность на матрице специльного вида, с параметрами $M = 4, n = 10, x = 1$ ($f_i = n \cdot \exp\dfrac{x}{i} \cdot \cos x$)

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), i \neq j, \\ (q_M - 1)^{i+j}, i = j \end{cases}$$

$$q_M = 1.001 - 2 \cdot M \cdot 10^{-3},\ i, j = \overline{1, n}$$

   Результат работы программы:
   Определитель: 0.2932
   Число обусловленности: 332.0582
   Корни и обратную матрицу см. в текстовом файле test4.txt

5. Тест 5:
   В данном тесте параметр $n = 100$

$$A_{ij} = \begin{cases} q_M^{i+j} + 0.1 \cdot (j - i), i \neq j, \\ (q_M - 1)^{i+j}, i = j \end{cases}$$

$$q_M = 1.001 - 2 \cdot M \cdot 10^{-3},\ i, j = \overline{1, n}$$

   Результат работы программы:
   Определитель: 0.0000
   Число обусловленности: 5504.4312
   Данная матрица была протестирована как на обычном, так и на модифицированном методе Гаусса, результаты см. в test5.txt

Выводы

Во время исследовательской работы было выявлено, что метод Гаусса вполне устойчив на матрицах небольшой размерности с хорошей обусловленностью. Однако на матрицах больших размерностей, и которые имеют большое число обусловленности, метод Гаусса вычисляет корни с ощутимой погрешностью. Модифицированный метод Гаусса так же имеет небольшую погрешность на таких матрицах, но она значительно меньше, чем в стандартном методе Гаусса.

Итерационные методы решения СЛАУ

Постановка задачи

Дана система уравнений $Ax = f$ порядка $n \times n$ с невырожденной матрицей A. Написать программу численного решения данной системы линейных алгебраических уравнений, использующую численный алгоритм итерационного метода верхней релаксации:

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

При $\omega = 1$(метод Зейделя) формула принимает вид:

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

Цели и задачи практической работы

1. Решить заданную СЛАУ итерационным методом верхней релаксации
2. Разработать критерий остановки итерационного процесса, гарантирующий получение приближенного решения исходной системы СЛАУ с заданной точностью
3. Изучить скорость сходимости итераций к точному решению задачи.
   Провести эксперименты с различными значениями итерационного параметра ω

Описание алгоритмов решения

Метод верхней релаксации

Запишем формулу

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

в покомпонентном виде, выразив x^{k+1}:

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}}\left(f_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i}^{n} a_{ij}x_j^k\right), \quad i = \overline{1,n}$$

Итак, получили итерационную покомпонентую формулу для вектора неизвестных. При этом, итерационный метод сходится, если выполнены условия теоремы Самарского: $A = A^* > 0$
Будем это учитывать и решать эквивалентную $Ax = f$ систему:

$$A^T Ax = A^T f$$

Критерий остановки

Чтобы разработать критерий остановки, рассмотрим следующие величины:

$$\text{Погрешность } z_k = x_k - x$$
$$\text{Невязка } \psi_k = Ax_k - f$$

Тогда получаем:

$$\psi_k = Ax_k - f = A(z_k + x) - f = Az_k => z_k = A^{-1}\psi_k => ||z_k|| <= ||A^{-1}|| \cdot ||\psi_k||$$

Получили, что в качестве критерия остановки будет служить следующее условие:

$$||A^{-1}|| \cdot ||\psi_k|| < \varepsilon$$

Описание программы

Добавим в уже описанный класс СЛАУ следующую функцию, реализующую метод верхней релаксации:

```
def over_relax(self, omega, epsilon, max_iter = 20000): # метод верхней релаксации
    transposed = transpose(self.A)
    A = dot_product(transposed,self.A) # A^TA
    f = dot_product(transposed, self.b) # A^Tf
    sl = Slau(list(A), f)
    sl.inverse() # посчитать обратную для A^TA матрицу

    cur = [0]*self.n
    # поставим ограничение в max_iter итераций
    count = 0
    while error(sl,cur) > epsilon and count < max_iter:
        prev = copy(cur)
        for i in range(self.n):
            s1 = 0
            for j in range(0, i):
                s1 += A[i][j]*cur[j]
            s2 = 0
            for j in range(i, self.n):
                s2 += A[i][j]*prev[j]
            # формула i-ой компоненты x_{k+1}
            cur[i] = prev[i] + ((f[i] - s1 - s2)*omega)/A[i][i]
        count += 1
    return cur, count
```

Кроме того, была написана следующая вспомогательная функция:

```
# Прогнать метод верхней релаксации для всех 0 < omega < 2 с шагом step
def research(sl, epsilon, step = 0.5, start = 0, stop = 2):
    my_range = [x*step for x in range(1 + int(start/step), int(stop/step))]

    for omega in my_range:
        cur, count = sl.over_relax(omega, epsilon)
        print('Omega = ', omega, ' Iter count: ', count)
```

Тестирование программы

1. Тест 11-1:
   Ниже приведена таблица соответствий параметра ω и количества итераций
   Число обусловленности: 30

$\varepsilon = 0.01,\ step = 0.1$	
ω	Количество итераций
0.1	4521
0.2	2143
0.3	1351
0.4	956
0.5	719
0.6	562
0.7	450
0.8	367
0.9	302
1.0	251
1.1	209
1.2	174
1.3	144
1.4	118
1.5	94
1.6	71
1.7	47
1.8	47
1.9	101

   Как видим, для матрицы из примера 11-1 быстрее всего сходится метод верхней релаксации с параметром 1.8

2. Тест 11-2:
   Матрица 11-2, изменённая(см. метод Гаусса).
   Число обусловленности: 45

$\varepsilon = 0.01,\ step = 0.05$	
ω	Количество итераций
0.05	7773
0.1	3776
0.15	2444
0.2	1778
0.25	1378
0.3	1111
0.35	921
0.4	778
0.45	667
0.5	578
0.6	445
0.65	394
0.7	349
0.75	311
0.8	278
0.85	248
0.9	221
0.95	197
1.0	174
1.05	151
1.1	127
1.15	99
1.2	74
1.25	117
1.3	131
1.35	135
1.4	139
1.45	186
1.5	195
1.55	199
1.6	254
1.65	263
1.7	325
1.75	393
1.8	524
1.85	667
1.9	1012
1.95	2049

   Для данной матрицы минимум количества итераций достигается при $\Omega \approx 1.2$

3. Тест 11-3:
   Матрица 11-3, изменённая.
   Число обусловленности: 310

$\varepsilon = 0.1$	
ω	Количество итераций
0.5	>20000
1.0	17076
1.5	6521
1.6	5024
1.7	3478
1.8	2486
1.9	4397
1.74	2703
1.75	2447
1.76	2154
1.77	1889
1.771	1868
1.772	1849
1.773	1833
1.774	1823
1.775	2494

   Как видно из таблицы выше, можно подобрать «идеальное» значение ω для матрицы с относительно плохой обусловленностью.
   В данном случае $\Omega \approx 1.774$

4. Тест 11-4 и 11-5:
   Матрица заданная по специальной формуле(см. стр. 11).
   Порядок $n = 10$
   Число обусловленности: 332

$\varepsilon = 0.1$	
ω	Количество итераций
0.5	20000
1.0	9060
1.5	2487
1.521	1795
1.522	1764
1.523	1735
1.524	1707
1.525	1680
1.526	1655
1.527	1631
1.528	1608
1.529	1588

   Для матриц такого вида оптимальным значением является $\Omega \approx 1.529$
   Отметим, что матрицы такого типа с порядком $n >> 1$ будут очень плохо обусловленны, так например, матрица порядка 100 имеет число обусловленности 5504. Итерационный процесс на таких матрицах будет сходиться довольно-таки долго, 20000 итераций не хватает, даже если брать оптимальное значение $\Omega = 1.529$
   Так же отмечу, что во всех тестах погрешность конечного ответа оказалась меньше, чем заданное число ε.

Выводы

Скорость сходимости метода верхней релаксации сильно зависит от выбора значения параметра ω, при этом оптимальное значение параметра необходимо подбирать для каждого типа матриц отдельно.
Для СЛАУ с хорошо обусловленными матрицами итерационный процесс довольно-таки быстро дает точное значение корней. Однако для плохо обусловленных матриц метод верхней релаксации сходится гораздо дольше метода Гаусса.