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 219 учебной группы факультета ВМК МГУ

Ратникова Тимофея Георгиевича

гор. Москва
2022 г.

Решение СЛАУ методом Гаусса и методом Гаусса с выбором главного элемента

Постановка задачи

Дана система уравнений Ax=f порядка n×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.

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 $MA=\|A\|\times\|A^{-1}\|$;
5) Исследовать вопрос вычислительной устойчивости метода Гаусса (при больших значениях параметра n);
6) Правильность решения СЛАУ подтвердить системой тестов

Алгоритмы решения

Решение СЛАУ методом Гаусса:

Сначала ищем строку с ненулевым первым элементом. Такая строка точно найдется в силу новорожденности матрицы A. Меняем эту строку местами с первой строкой. Разделим все элементы этой строки и f_1 на ненулевой a_{11}. Вычтем из последующих n-1 строк первую строку, домноженную на первый элемент соответствующей строки. Повторим такие же действия для n-1 строк. Таким образом мы получили верхнетреугольную матрицу. Далее происходит обратный ход — мы идем снизу вверх по матрице, вычисляя корни уравнений.

Решение СЛАУ методом Гаусса с выбором главного элемента:

Отличается от стандартного метода Гаусса только тем, что сначала выбирается строка с наибольшим по модулю ведущим элементом.

Вычисление определителя матрицы:

Приводим матрицу к верхнетреугольной форме. Определитель равен произведению диагональных элементов матрицы и $(-1)^k$, где k – число перестановок строк/столбцов в методе Гаусса.

Вычисление обратной матрицы:

Присоединяем к данной матрице единичную матрицу и применяем метод Гаусса, тем самым зануляя все элементы ниже главной диагонали. Потом аналогично зануляем все элементы выше главной диагонали. В результате получим единичную матрицу. А присоединенная единичная матрица приведется к обратной матрице изначальной.

Текст программы

```
import copy
import numpy as np

def fill_matrix1(a, b, file_name): #Заполнение матрицы из файла
    file = open(file_name, 'r')
    f = file.readlines()
    n = int(f[0])
    for i in range(1, n+1):
        a.append(f[i])
        a[i - 1] = a[i - 1].split(" ")
        for j in range(n - 1):
            a[i - 1][j] = int(a[i - 1][j])
        a[i - 1][n - 1] = int(a[i - 1][n - 1][:len(a[i - 1][n - 1]) - 1])
    f[n + 1] = f[n + 1].split(" ")
    for i in range(n):
        b.append(int(f[n + 1][i]))
    return n

def fill_matrix2(a, b):
    print("Enter matrix size")
    n = int(input())
    for i in range(n):
        print("Enter the", i + 1, "row of the matrix")
        a.append([])
        a[i] = list(map(int,input().split()))
    print("Enter the vector of solutions")
    s = input().split(" ")
    for i in range(n):
        b.append(int(s[i]))
    return n

def fill_formula(a, b):
    n = 30
    m = 9
    for i in range(n):
        a.append([])
        for j in range(n):
            if i != j:
                a[i].append((i + j + 2)/(m + n))
            else:
                a[i].append(n + m**2 + (j + 1)/m + (i + 1)/n)
    for i in range(1, n+1):
        b.append(i**2 - 100)

def swap_lines(a, ind1, ind2):#Функция меняющая строки в матрице местами
    a[ind1], a[ind2] = a[ind2], a[ind1]

def gauss_solution(a, b, c, n): #Решение СЛАУ методом Гаусса
    for i in range(0, n - 1): #Приводим к верхней треугольной форме
        for j in range(i + 1, n):
            if a[i][i] == 0:
                line = i
                for k in range(i + 1, n):
                    if (a[k][i] != 0):
                        line = k
                        break
                swap_lines(a, i, line)
```

```
                swap_lines(b, i, line)
            fact = a[j][i] / a[i][i]
            for k in range(i, n):
                a[j][k] -= fact * a[i][k]
            b[j] -= b[i] * fact
    #Ищем решения
    if a[n - 1][n - 1] == 0:
        c[n - 1] = 0
    else:
        c[n - 1] = b[n - 1]/a[n-1][n-1]

    for i in range(n - 2, -1, -1):
        for j in range(n - 1, i, -1):
            b[i] -= a[i][j] * c[j]
            if b[i] == 0:
                c[i] = 0
            else:
                c[i] = b[i] / a[i][i]

def modified_gauss(a, b, c, n): #Решение СЛАУ методом Гаусса с выделеением
главного элемента
    res = [0] * n
    res_num = list()
    for i in range(n):
        res_num.append(i)
    for i in range(n - 1):
        max_elem = abs(a[i][i])
        ind = i
        for k in range(i + 1, n):
            if (abs(a[i][k]) > max_elem):
                ind = k
                max_elem = abs(a[i][k])
        if i != ind:
            for k in range(0, n):
                swap_lines(a, i, ind)
                swap_lines(b, i, ind)
        res_num[i], res_num[ind] = res_num[ind], res_num[i]
        fact = 0
        for j in range(i + 1, n):
            fact = a[j][i]/a[i][i]
            for k in range(i, n):
                a[j][k] -= fact * a[i][k]
            b[j] -= fact * b[i]
    if a[n - 1][n -1] == 0:
        res[n - 1] = 0
    else:
        res[n - 1] = b[n - 1]/a[n - 1][n -1]
    for i in range(n - 2, -1, -1):
        fact = b[i]
        for j in range(n - 1, i, -1):
            fact -= a[i][j] * res[j]

        if fact == 0:
            res[i] = 0
        else:
            res[i] = fact / a[i][i]
    for i in range(n):
        if res_num[i] != i:
            res[i], res[res_num[i]] = res[res_num[i]], res[i]
```

```
            res_num[i], res_num[res_num[i]] = res_num[res_num[i]],
res_num[i]
    for i in range(n):
        c[i] = res[i]
def det(a, n): # Функция считает определитель матрицы
    a2 = copy.deepcopy(a)
    sign = 1
    for i in range(0, n - 1): #Приводим к верхней треугольной форме
        for j in range(i + 1, n):
            if a2[i][i] == 0:
                line = i
                for k in range(i + 1, n):
                    if (a2[k][i] != 0):
                        line = k
                        break
                swap_lines(a2, i, line)
                sign *= -1
            fact = a2[j][i] / a2[i][i]
            for k in range(i, n):
                a2[j][k] -= fact * a2[i][k]
    ans = 1
    for i in range(n):
        ans *= a2[i][i]
    ans *= sign
    return ans

def inverse_matrix(a, n): # Поиск обратной матрицы
    a2 = copy.deepcopy(a)
    for i in range(n):
        for j in range(n):
            if j == i:
                a[i][j] = 1
            else:
                a[i][j] = 0
    for j in range(n):
        t = j
        while t < n and a2[t][j] == 0:
            t += 1
        if t != j:
            a2[j], a2[t] = a2[t], a2[j]
            a[j], a[t] = a[t], a[j]
        fact = a2[j][j]
        for k in range(n):
            a2[j][k] = a2[j][k] / fact
            a[j][k] = a[j][k] / fact
        for i in range(t + 1, n):
            if (a2[i][j] != 0):
                fact = a2[i][j]
                for k in range(n):
                    a2[i][k] -= a2[j][k] * fact
                    a[i][k] -= a[j][k] * fact
    for j in range(n - 1, 0, -1):
        for i in range(j - 1, -1, -1):
            for k in range(n):
                a[i][k] -= a2[i][j] * a[j][k]

def matrix_norm(a, n): # Первая матричная норма
    ans = 0
    for i in range(n):
        summ = 0
```

```
            for j in range(n):
                summ += abs(a[i][j])
            if summ > ans:
                ans = summ
        return ans

def conditionality_num(a, n):
    return matrix_norm(a, n) * matrix_norm(inverse_matrix(a, n), n)

def relaxation(a, b, n, max_iters, omega, eps):
    c = np.zeros(n)
    counter = 0
    while counter < max_iters:
        c_new = np.copy(c)
        for i in range(n):
            s1 = 0
            s2 = 0
            for j in range(i):
                s1 += a[i][j] * c_new[j]
            for j in range(i, n):
                s2 += a[i][j] * c[j]
            c_new[i] = c[i] + ((b[i] - s1 - s2) * omega) / a[i][i]
        s = 0
        for i in range(n):
            s += (c_new[i] - c[i]) ** 2
        if np.sqrt(s) <= eps:
            break
        c = c_new
        counter += 1
    return c


print("Enter 1 to fill the matrix from the keyboard")
print("or 2 to fill the matrix from the file")
print("or another number to fill by formula")
mode = int(input())
a = list()
b = list()
if mode == 1:
    fill_matrix2(a, b)
elif mode == 2:
    print("Enter file name")
    f = input()
    f = "input.txt"
    fill_matrix1(a, b, f)
else:
    fill_formula(a, b)
```

Тестирование

Тестируемые системы — системы из п. 1.9:
1)

Входные данные:
4
2 -5 3 1 5
3 -7 3 -1 -1
5 -9 6 2 7
4 -6 3 1 8

Решения методом Гаусса:
x1 = 0
x2 = -3
x3 = -5.333
x4 = 6

Определитель:
18

Обратная матрица:
-1.000 -0.000 0.333 0.333
-1.000 -0.000 0.667 -0.333
-1.000 0.167 1.056 -0.944
1.000 -0.500 -0.500 0.500
Число обусловленности:
67

Проверка с помощью WolframAlpha:
Корни: 0, -3, -16/3, 6

2)
Входные данные:
4
4 3 -9 1 9
2 5 -8 -1 8
2 16 -14 2 24
2 3 -5 -11 7

Решения методом Гаусса:
x1 = 3
x2 = 2
x3 = 1

x4 = 0

Определитель:
-868

Обратная матрица:
0.689 -1.364 0.253 0.233
0.118 -0.507 0.182 0.090
0.240 -0.779 0.175 0.124
0.048 -0.032 0.016 -0.081

Число обусловленности:
86.331

Проверка с помощью WolframAlpha:
Корни: 3, 2, 1, 0

3)
Входные данные:
4
12 14 -15 24 5
16 18 -22 29 8
18 20 -21 32 9
10 12 -16 20 4

Решения методом Гаусса:
x1 = 2.222
x2 = -1.666
x3 = -0.111
x4 = 0

Определитель:
36

Обратная матрица:
1.000 2.889 -1.667 -2.722
-2.500 -4.667 3.500 4.167
0.000 -0.444 0.333 0.111
1.000 1.000 -1.000 -1.000

Число обусловленности:
1349.833

Проверка с помощью WolframAlpha:

Корни: 20/9, -5/3, -1/9, 0

4)
Матрица из приложения 2:

$$A_{ij} = \begin{cases} \dfrac{i+j}{m+n}, & i \neq j, \\ n + m^2 + \dfrac{j}{m} + \dfrac{i}{n}, & i = j, \end{cases}$$

где $i, j = 1,...n$.

n	m	b_i , $i=1, ..., n$
5	9	$b_i = i^2 - 100$

Матрица:
86.311 0.214 0.286 0.357 0.429 | -99
0.214 86.622 0.357 0.429 0.500 | -96
0.286 0.357 86.933 0.500 0.571 | -91
0.357 0.429 0.500 87.244 0.643 | -84
0.429 0.500 0.571 0.643 87.556 | -75

Решение методом Гаусса:
x1 = -1.133
x2 = -1.092
x3 = -1.028
x4 = -0.941
x5 = -0.831

Определитель:
4963533290.324

Обратная матрица:
0.012 -0.000 -0.000 -0.000 -0.000
-0.000 0.012 -0.000 -0.000 -0.000
-0.000 -0.000 0.012 -0.000 -0.000
-0.000 -0.000 -0.000 0.011 -0.000
-0.000 -0.000 -0.000 -0.000 0.011

Число обусловленности:
1.054

Проверка с помощью WolframAlpha:
Корни: -1.132, -1.092, -1.027, -0.941 -0.831

5) Для матрицы 30*30 программа также была, протестирована, но из-за больших объемов данных проблематично предоставить результаты теста в отчете.

Вывод:

Были реализованы метод Гаусса и метод Гаусса с выбором ведущего элемента. А также вспомогательные функции поиска определителя, обратной матрицы, числа обусловленности. Было выявлено, что данный метод вполне устойчив на матрицах небольшого размера, хорошим числом обусловленности. Но на матрицах большого размера с большим числом обусловленности, метод Гаусса вычисляет корни с ощутимой погрешностью. При этом погрешность у метода Гаусса с выбором ведущего элемента значительно меньше, чем у стандартного варианта.

ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Постановка задачи

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

в случае использования итерационного метода верхней релаксации итерационный процесс имеет следующий вид:

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

где $D, A^{(-)}$ - соответственно диагональная и нижняя треугольные матрицы, k - номер текущей итерации;

где ω - итерационный параметр, в случае если он равен 1, то данный метод переходит в метод Зейделя.

Цели и задачи практической работы

1) Решить заданную СЛАУ итерационным методом (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 (при использовании итерационного метода верхней релаксации провести эксперименты с различными значениями итерационного параметра ω

Алгоритмы решения

Данный метод вычисляет приближенные решения СЛАУ по формуле:

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

где *D,* A, - соответственно диагональная и нижняя треугольные матрицы, *k* - номер текущей итерации, *w* - итерационный параметр (при *w* =1 метод верхней релаксации переходит в метод Зейделя). В данном методе результат постепенно приближается к искомым корням уравнения, поэтому когда длина вектора невязки станет меньше заданной рогрешности, метод остановится.

Описание программы

Код программы был описан выше на страницах 4-7.

Тестирование

Все матрицы для тестов аналогичны матрицам из тестов для метода Гаусса, функция осуществляющая метод релаксации запускается с точностью 0.0001.

1)Для примеров 1-3 метод релаксации неприменим.

2)(Матрица 5 на 5, 4 пример из первых тестов)

w = 0.1 iterations = 73
w = 0.2 iterations = 38
w = 0.3 iterations = 25
w = 0.4 iterations = 18
w = 0.5 iterations = 14
w = 0.6 iterations = 11
w = 0.7 iterations = 8
w = 0.8 iterations = 6
w = 0.9 iterations = 5
w = 1.0 iterations = 3
w = 1.1 iterations = 5
w = 1.2 iterations = 7
w = 1.3 iterations = 9
w = 1.4 iterations = 12
w = 1.5 iterations = 16
w = 1.6 iterations = 21
w = 1.7 iterations = 30
w = 1.8 iterations = 48
w = 1.9 iterations = 102

Best result with w = 1.0
Number of iterations = 3

3)Входные данные:

4
13 1 2 3 51
2 19 9 10 43
4 12 17 11 17
5 5 5 55 20

w = 0.1 iterations = 170
w = 0.2 iterations = 88
w = 0.3 iterations = 59
w = 0.4 iterations = 43

w = 0.5 iterations = 33
w = 0.6 iterations = 26
w = 0.7 iterations = 20
w = 0.8 iterations = 16
w = 0.9 iterations = 13
w = 1.0 iterations = 9
w = 1.1 iterations = 6
w = 1.2 iterations = 8
w = 1.3 iterations = 10
w = 1.4 iterations = 13
w = 1.5 iterations = 18
w = 1.6 iterations = 23
w = 1.7 iterations = 33
w = 1.8 iterations = 51
w = 1.9 iterations = 105

Best result with w = 1.1
Number of iterations = 6

4)Матрица 30 на 30 из 5 теста, заданная формулой на странице 10

w = 0.1 iterations = 91
w = 0.2 iterations = 46
w = 0.3 iterations = 30
w = 0.4 iterations = 22
w = 0.5 iterations = 17
w = 0.6 iterations = 13
w = 0.7 iterations = 10
w = 0.8 iterations = 8
w = 0.9 iterations = 6
w = 1.0 iterations = 5
w = 1.1 iterations = 7
w = 1.2 iterations = 9
w = 1.3 iterations = 12
w = 1.4 iterations = 16
w = 1.5 iterations = 20
w = 1.6 iterations = 27
w = 1.7 iterations = 39
w = 1.8 iterations = 61
w = 1.9 iterations = 130

Best result with w = 1.0
Number of iterations = 5

Выводы

Был реализован и протестирован метод верхней релаксации. В результате тестирования были найдены лучшие параметры w для тестируемых матриц, которые давали наибольшую скорость сходимости. Также было выявлено, что скорость сходимости метода верхней релаксации существенно зависит от выбора w.