Московский Государсвенный 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 204 учебной группы факультета ВМК МГУ
Воробьева Сергея Юрьевича

Москва, 2020

Оглавление

Глава 1

Введение

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;
- определителя заданных матриц;
- матрицы, обратной данной.

Кроме того, в работе исследуются вопрос устойчивости метода Гаусса при больших размерах матрицы коэффициентов и вопрос скорости сходимости к точному решению задачи итераций метода верхней релаксации при изменении значения итерационного параметра ω.

Глава 2

Алгоритмы решения

2.1 Метод Гаусса

Решения системы линейных алгебраических уравнений методом Гаусса производится в два этапа, именуемые "прямым"и "обратным"ходом, в результате которых из соответствующей исследуемой системе линейных алгебраических уравнений расширенной матрицы находится решение системы. В результате "прямого хода"матрица коэффициентов, входящая в расширенную матрицу, путем линейных пребразований 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 отличие от классического метода Гаусса в методе Гаусса с выбором главного элемента на каждой итерации выбирается максимальный из всех элементов матрицы, что позволяет уменьшить погрешность вычислений. Более подробно оба метода описаны в [1].

Задача вычисления определителя сводится к задаче приведения исследуемой матрицы к верхнему треугольному виду и последующим вычислением произведения её диагональных элементов. Для приведения матрицы к диагональному виду используется классический метод Гаусса. Заметим, что строки матрицы при этом местами не переставляются.

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. Как и при нахождении определителя матрицы, строки расширенной матрицы местами не перставляются. Теоретическое обоснование метода Гаусса-Жордана может быть найдено в [2].

2.2 Метод верхней релаксации

Метод вехний релакцации является стационарным итерационным методом, в котором каждый следующий вектор приближения точного решения вычисляется по формуле:

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

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,

$T_{\text{н}}$ - нижняя диагональная матрица матрицы A,

D - матрица диагональных элементов матрицы A.

По теореме Самарского [1] для положительно определенных матриц A метод верхней релаксации сходится, если $0 < \omega < 2$. Однако, оптимальное значение ω в каждом отдельном случае подбирается экспериментально и зависит от матрицы A. В данной реализации метода допускается возможность изменения итерационного параметра. Значением по умолчанию является значение $\omega = \frac{4}{3}$. За вектор начального приближения берется нулевой вектор.

Глава 3

Программа

В качестве языка программирования был выбран Python3 в виду его гибкости и простоты синтаксиса. Каждый численный метод реализован отдельной функцией, которые представлены ниже. В качестве вспомогательного модуля использовался модуль NumPy. Его структуры данных позволяют удобно хранить и преобразовывать матрицы и вектора. Для проверки правильности работы программы использовался модуль NumPy.linalg, в частности, метод solve(A, b)

```
def det_inverse(matrix, n):
    # Определитель и инвертирование матрицы
    matrix = np.array(matrix, dtype=np.float64)
    det = 1.0
    inv_matrix = matrix[:, :n]
    E = np.identity(n) # Единичная матрица
    for i in range(0, n):
        inv_matrix = np.column_stack((inv_matrix, E[:, i])) # Присоединяем ее
    for i in range(0, n): # "Левую" матрицу приводим к нижнетреугольному виду
        if inv_matrix[i][i] == 0:
            for j in range(i + 1, n):
                if inv_matrix[j][i] != 0:
                    inv_matrix[[i, j]] = inv_matrix[[j, i]] # Меняем строчки местами
                    break
        det *= inv_matrix[i][i] # Последовательно вычисляем определитель
        if det == 0:
            return 0, np.nan
        inv_matrix[i] /= inv_matrix[i][i]
        for j in range(i + 1, n):
            inv_matrix[j] -= inv_matrix[i] * inv_matrix[j][i]
    for i in range(n - 1, -1, -1):
        for j in range(i - 1, -1, -1):
            inv_matrix[j] -= inv_matrix[i] * inv_matrix[j][i]
    inv_mat = inv_matrix[:, n:]
    return det, inv_mat
```

```
def mat_norm(matrix, n):
    print(matrix)
    matrix = np.array(matrix, dtype=np.float64)
    max = 0
    for i in range(0, n):
        max_i = 0
        for j in range(0, n):
            max_i += abs(matrix[i][j])
        if max_i > max:
            max = max_i
    return max
```

```python
def Gauss(matrix, n):
    # Прямой ход
    matrix = np.array(matrix, dtype=np.float64)
    for i in range(0, n):
        if matrix[i][i] == 0: # Если диагональный элемент нулевой
            for j in range(i + 1, n): # Ищем строку с matrix[j][i] != 0
                if matrix[j][i] != 0:
                    matrix[[i, j]] = matrix[[j , i]] # Меняем местами строчки
                    break
        matrix[i] /= matrix[i][i] # Делим i-ю строчку на ее i-й элемент
        for j in range(i + 1, n):
            matrix[j] -= matrix[i] * matrix[j][i] # Вычитаем из каждой строки i-ю,
                                                  # умноженную на нужный коэф.
    # Обратный ход
    x = matrix[:, n] # Вектор неизвестных со значениями свободных членов
    for i in range(n - 1, -1, -1):
        for j in range(n - 1, i, -1):
            x[i] -= x[j] * matrix[i][j] # Восстанавливаем ответ
    return x
```

```python
def GaussDiag(matrix, n):
    # Прямой ход
    matrix = np.array(matrix, dtype=np.float64)
    permutations = np.array(range(0, n))
    for i in range(0, n):
        if matrix[i][i] == 0: # Если диагональный элемент нулевой
            for j in range(i + 1, n): # Ищем строку с matrix[j][i] != 0
                if matrix[j][i] != 0:
                    matrix[[i, j]] = matrix[[j , i]] # Меняем местами строчки
                    break
        # Модифицированный метод гаусса
        col_max = i + np.argmax(abs(matrix[i][i:n]))
        matrix[:, [i, col_max]] = matrix[:, [col_max, i]] # Меняем столбцы местами
        permutations[i], permutations[col_max] = permutations[col_max], permutations[i]

        matrix[i] /= matrix[i][i] # Делим i-ю строчку на ее i-й элемент
        for j in range(i + 1, n):
            matrix[j] -= matrix[i] * matrix[j][i]

    # Обратный ход
    x = matrix[:, n] # Вектор неизвестных со значениями свободных членов
    for i in range(n - 1, -1, -1):
        for j in range(n - 1, i, -1):
            x[i] -= x[j] * matrix[i][j] # Восстанавливаем ответ
    res = np.array(list(zip(x, permutations))) # Восстанавливаем порядок в векторе
    res = res[res[:, 1].argsort()]
    x = res[:, 0] # Берем только значения
    return x
```

```
def iter_method(data, w, n, EPS=1e-2):
    f = data[:, n]
    cur_x = np.zeros(n)
    f_sum = 0
    for i in range(0, n):
        f_sum += f[i] ** 2
    nev_norm = sqrt(f_sum)
    _, a_inv = det_inverse(data, n)
    a_inv_norm = mat_norm(a_inv, n)
    iter_count = 0
    flag = True
    while flag == True: # Основной цикл
        prev_x = np.copy(cur_x)
        flag = False
        mem = np.copy(prev_x)
        for i in range(0, n):
            sum1 = 0
            for j in range(0, i):
                sum1 += data[i][j] * cur_x[j]
            sum2 = 0
            for j in range(i, n):
                sum2 += data[i][j] * prev_x[j]
            cur_x[i] = f[i] - sum1 - sum2
            cur_x[i] *= w
            cur_x[i] /= data[i][i]
            cur_x[i] += prev_x[i]
        nev = np.matmul(data[:, :n], cur_x) - f
        nev_norm = 0
        for i in range(0, n):
            nev_norm += nev[i] ** 2
        nev_norm = sqrt(nev_norm)
        norm = a_inv_norm * nev_norm
        if norm >= EPS: # Продолжаем, если расстояние между решениями больше EPS
            flag = True
            prev_x = cur_x
            iter_count += 1
    return cur_x, iter_count
```

Глава 4

Тестирование

Тестирование правильности решения систем линейных алгебраических уравнений программой проводилось как с помощью проверки результатов работы программы путем подстановки их в исходную систему, так и с помощью сравнения результатов работы разных методов с решением через функцию $numpy.linalg.solve()$. Тестирование нахождения обратных матриц и определителя проверялось вручную для матриц, размером меньше 3×3, и с помощью сайта $WolframAlpha$ для матриц большего порядка. Первые три теста взяты из варианта №5 тестовых заданий. Тесты 5, 6 взяты из вариантов 1, 2 приложения 2. Тесты 6 и 7 придуманы для подробного тестирования итерационного метода

Тест 1	Результат
Matrix	[7 9 4 2 2] [2 -2 1 1 6] [5 6 3 2 3] [2 3 1 1 0]
Det	-5.000000000000005
Inverse	[1. 0.6 -2. 1.4] [-0. -0.2 -0. 0.2] [-1. -0.6 3. -3.4] [-1. 0. 1. 1.]
Gauss	[-0.4 -1.2 3.4 1.]
Gauss Diag	[-0.4 -1.2 3.4 1.]
Iter Method	[-inf -inf inf -inf]
NumPy	[-0.4 -1.2 3.4 1.]

Тест 2	Результат
Matrix	[6 3 2 3 5] [4 2 1 2 4] [4 2 3 2 0] [2 1 7 3 1]
Det	0
Inverse	-
Gauss	-
Gauss Diag	-
Iter Method	-
NumPy	-

Тест 3	Результат
Matrix	[1 3 2 -1 0] [2 -1 3 0 0] [3 5 -4 15 0] [1 17 4 0 0]
Det	330.0
Inverse	[2.5 -0.733 0.167 -0.533] [0.227 -0.139 0.015 0.006] [-1.591 0.776 -0.106 0.358] [-1. 0.4 0. 0.2]
Gauss	[0. 0. 0. 0.]
Gauss Diag	[0. 0. 0. 0.]
Iter Method	[-inf -inf inf -inf]
NumPy	[0. 0. 0. 0.]

Тест 4	Результат
Matrix	[1.40000e+02 2.00000e-02 ...] [2.00000e-02 1.40125e+02 ...] ... [7.80000e-01 8.00000e-01 ...]
Det	1.360757180518628e+86
Inverse	[0.007 0. ...] [0. 0.007 ...] ... [-0. -0. ...]
Gauss	[-0.61254795 -0.351742 ...
Gauss Diag	[-0.61254795 -0.351742 ...
Iter Method	[-inf -inf inf -inf] ...
NumPy	[-0.61254795 -0.351742 ...

Тест 5	Результат
Matrix	[8.400000e+01 3.571427e-02 ...] ... [6.785714e-01 7.142857e-01 ...]
Det	4.464076823137942e+38
Inverse	[0.012 -0. ...] ... [-0. -0. ...]
Gauss	[1.69124408 2.22672642 ...
Gauss Diag	[1.69124408 2.22672642 ...
Iter Method	[-inf -inf inf -inf ...
NumPy	[1.69124408 2.22672642 ...

Тест 6	Результат
Matrix	[3 -1 1 3] [1 1 1 5] [4 -1 4 5]
w	1
Iter Method	[1.9999e+00, 3.0000e+00, 1.8866e-05]
Iter Count	26
NumPy	[2. 3. -0.]

Тест 7	Результат
Matrix	[2 -1 1 6] [1 2 -1 -1] [1 -1 2 11]
w	4/3
Iter Method	[1.0000131 , 2.00000781, 6.00000142]
Iter Count	16
NumPy	[1. 2. 6.]

К сожалению, итерационный метод сошелся только на последних двух тестах. В целом, при $\omega = 4/3$ метод релаксации получал решение, достаточно близкое к точному менее, чем за 100 итераций. В тесте 6 для ускорения сходимости значение итерационного параметра было изменено на $\omega = 1$. Таким образом, в тесте 6 решение было найдено с помощью метода Зейделя.

Глава 5

Заключение

В ходе работы мной были освоены вычислительные методы решения систем линейных алгебраических уравнений, а именно метод верхней релаксации и метод Гаусса. В ходе тестирования стало очевидно, что ввиду ограниченной применимости метода верхней релаксации, порой он не дает хороших результатов. Тем не менее, при аккуратном подборе итерационного параметра, он может оказаться значительно быстрей метода Гаусса. Преимуществом последнего же является простота реализиции и широкая применимость в задачах, связанных с исследованием матриц, в том числе нахождением обратной матрицы и детерминанта. Для плохо обусловленных матриц и матриц достаточно большого порядка метод является вычислительно неустойчивым, что продемонстрировало бы различие в решениях, найденных обычным методом Гаусса и методом Гаусса с выбором главного элемента.

Литература

[1] Самарский А.А. Введение в численные методы. СПб., 2009.

[2] Белоусов И.В. Матрицы и определители. Кишинев, 2006.