ИНСТИТУТ ТРАНСПОРТА И СВЯЗИ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕЛЕКОММУНИКАЦИЙ

Лабораторная работа №1
По дисциплине
«Численные методы и прикладное программирование»

Тема:
«Методы решения системы линейных уравнений.
Число обусловленности матрицы»

Работу выполнили:

Дзенис Ричард
Кобелев Денис
Якушин Владислав

Рига
2017 г.

Содержание

1 Формулировка задания

- Реализовать программным путём метод исключения Гаусса и итерационный метод Гаусса-Зейделя.
- Результат работы программы проверить с помощью предоставленных примеров.
- Ручным или программным путём рассчитать число обусловленности матриц для примеров (3) и (5).
- Для расчёта обусловленности выбрать Манхэттенскую или Евклидову норму.
- Составить отчёт с результатами вычислений и выводами, содержащими сравнение двух реализованных методов, а так же объяснить значения полученный при вычислении числа обусловленности матриц.

1.1 Примеры

$$\begin{cases} x_1 - 2x_2 + x_3 = 2 \\ 2x_1 - 5x_2 - x_3 = -1 \\ -7x_1 + x_3 = -2 \end{cases} \tag{1}$$

$$\begin{cases} 5x_1 - 5x_2 - 3x_3 + 4x_4 = -11 \\ x_1 - 4x_2 + 6x_3 - 4x_4 = -10 \\ -2x_1 - 5x_2 + 4x_3 - 5x_4 = -12 \\ -3x_1 - 3x_2 + 5x_3 - 5x_4 = 8 \end{cases} \tag{2}$$

$$\begin{cases} 2x_1 - x_2 - x_3 = 5 \\ x_1 + 3x_2 - 2x_3 = 7 \\ x_1 + 2x_2 + 3x_3 = 10 \end{cases} \tag{3}$$

$$\begin{cases} 8x_1 + 5x_2 + 3x_3 = 30 \\ -2x_1 + 8x_2 + x_3 = 15 \\ x_1 + 3x_2 - 10x_3 = 42 \end{cases} \tag{4}$$

$$\begin{cases} 0.78x_1 + 0.563x_2 = 0.217 \\ 0.913x_1 + 0.659x_2 = 0.254 \end{cases} \tag{5}$$

2 Метод исключения Гаусса с ведущим элементом

2.1 Листинг

```
// Input: linear system ‘Ax=b‘,
//        where Ab is matrix A combined with vector b.
//        Size of Ab is (n, n+1).
// Output: x.
void solve(double **Ab, ssize_t n, double *x) {
    ssize_t base, r, c;
    double sum;

    // Forward elimination
    for (base = 0; base < n; base++) {
        // Select maximal element in the column;
        // optimize by skipping base row.
        c = base;
        size_t leading_row = base;
        double max_value = Ab[base][base];
        for (r = base + 1; r < n; r++) {
            double abs_val = fabs(Ab[r][c]);
            if (abs_val > max_value) {
```

```
                leading_row = r;
                max_value = abs_val;
            }
        }

        // Swap base row with with leading row
        std::swap(Ab[base], Ab[leading_row]);

        // Eliminate base column
        for (r = base + 1; r < n; r++) {
            double coef = Ab[r][base] / Ab[base][base];
            for (c = base; c <= n; c++) { // including vector B
                Ab[r][c] -= coef * Ab[base][c];
            }
        }
    }

    // Backward substitution
    for (base = n - 1; base >= 0; base--) {
        sum = 0.0;
        for (c = base + 1; c < n; c++) {
            sum += Ab[base][c] * x[c];
        }
        x[base] = (Ab[base][n] - sum) / Ab[base][base];
    }
}
```

2.2 Результаты работы алгоритма

2.2.1 Результат работы алгоритма на примере (1)

```
        0.52            0.08            1.64
```

2.2.2 Результат работы алгоритма на примере (2)

```
    -12.8235        -2.29412         11.7647         19.2353
```

2.2.3 Результат работы алгоритма на примере (3)

```
      3.8125          1.6875          0.9375
```

2.2.4 Результат работы алгоритма на примере (4)

```
           3               3              -3
```

2.2.5 Результат работы алгоритма на примере (5)

```
           1              -1
```

3 Метод Гаусса-Зейделя

3.1 Листинг

```
bool solve(double **Ab, ssize_t n, double *x, double eps) {
    int iteration = 0;
    ssize_t i, j;
    double acc, prev_acc = HUGE_VALF;
    do {
        std::cout << "I = " << iteration << std::endl;
        acc = 0.0f;
        for (i = 0; i < n; i++) {
            double denom = Ab[i][i];
            double new_xi = Ab[i][n] / denom;
```

```
            for (j = 0; j < n; j++) {
                if (i == j)
                    continue;
                new_xi -= Ab[i][j] / denom * x[j];
            }
            acc = std::fmaxf(acc, fabs(new_xi - x[i]));
            x[i] = new_xi;
        }
        std::cout << "X = ";
        for (auto i = 0; i < n; i++) {
            std::cout << std::setw(16) << x[i];
        }
        std::cout << std::endl;
        std::cout << "ACC = " << acc << std::endl;
        if (acc >= prev_acc)
            return false;
        prev_acc = acc;
        iteration++;
    } while (acc > eps);
    return true;
}
```

3.2 Результаты работы алгоритма с точностью 10^{-3}

3.2.1 Результат работы алгоритма на примере (1)

```
I = 0
X =                2                1               12
ACC = 12
I = 1
X =               -8             -5.4              -58
ACC = 70
does not converge
```

3.2.2 Результат работы алгоритма на примере (2)

```
I = 0
X =             -2.2             1.95          -1.6625          -3.1125
ACC = 3.1125
I = 1
X =           1.2425          3.42938         -1.98266         -6.38578
ACC = 3.4425
does not converge
```

В первых двух примерах не выполняется условие сходимости для итерационного метода, что и было успешно обнаружено программой.

3.2.3 Результат работы алгоритма на примере (3)

```
I = 0
X =              2.5              1.5              1.5
ACC = 2.5
I = 1
X =                4                2         0.666667
ACC = 1.5
I = 2
X =          3.83333              1.5          1.05556
ACC = 0.5
I = 3
X =          3.77778          1.77778         0.888889
ACC = 0.277778
I = 4
```

```
X =            3.83333          1.64815          0.95679
ACC = 0.12963
I = 5
X =            3.80247           1.7037         0.930041
ACC = 0.0555556
I = 6
X =            3.81687          1.68107         0.940329
ACC = 0.0226337
I = 7
X =             3.8107          1.68999         0.936443
ACC = 0.00891632
I = 8
X =            3.81321          1.68656         0.937891
ACC = 0.00342936
I = 9
X =            3.81222          1.68785         0.937357
ACC = 0.00129553
I = 10
X =             3.8126          1.68737         0.937552
ACC = 0.00048265
          3.8126          1.68737         0.937552
```

3.2.4 Результат работы алгоритма на примере (4)

```
I = 0
X =               3.75           2.8125         -2.98125
ACC = 3.75
I = 1
X =            3.11016           3.0252         -2.98143
ACC = 0.639844
I = 2
X =            2.97729            2.992         -3.00467
ACC = 0.132869
I = 3
X =            3.00675          3.00227         -2.99864
ACC = 0.029464
I = 4
X =            2.99807          2.99935         -3.00039
ACC = 0.00868027
I = 5
X =            3.00055          3.00019         -2.99989
ACC = 0.00248163
I = 6
X =            2.99984          2.99995         -3.00003
ACC = 0.000711488
         2.99984          2.99995         -3.00003
```

3.2.5 Результат работы алгоритма на примере (5)

```
I = 0
X =           0.278205     -1.94545e-06
ACC = 0.278205
I = 1
X =           0.278207      -3.8909e-06
ACC = 1.94545e-06
        0.278207     -3.8909e-06
```

4 Определение невязки

```
X = solve(A, B)
print("X:", X.T)
print("Residual:", (B - A * X).T)
```

4.1 Полученные результаты для метода исключения Гаусса

4.1.0.1 Пример (1)

```
X: [[ 0.52  0.08  1.64]]
Residual: [[  0.00000000e+00  -1.11022302e-16   0.00000000e+00]]
```

4.1.0.2 Пример (2)

```
X: [[-12.8235   -2.29412  11.7647   19.2353 ]]
Residual: [[ -2.00000000e-04   2.00000000e-05   1.00000000e-04   1.40000000e-04]]
```

4.1.0.3 Пример (3)

```
X: [[ 3.8125  1.6875  0.9375]]
Residual: [[ 0.  0.  0.]]
```

4.1.0.4 Пример (4)

```
X: [[ 3  3 -3]]
Residual: [[0 0 0]]
```

4.1.0.5 Пример (5)

```
X: [[ 1 -1]]
Residual: [[ -8.32667268e-17   0.00000000e+00]]
```

4.2 Полученные результаты для метода Гаусса-Зейделя

4.2.0.1 Пример (3)

```
X: [[ 3.8126    1.68737   0.937552]]
Residual: [[ -2.78000000e-04   3.94000000e-04   4.00000000e-06]]
```

4.2.0.2 Пример (4)

```
X: [[ 2.99984  2.99995 -3.00003]]
Residual: [[  1.62000000e-03   1.10000000e-04   1.00000000e-05]]
```

4.2.0.3 Пример (5)

```
X: [[  2.78207000e-01  -3.89090000e-06]]
Residual: [[  7.30576700e-07  -4.26896900e-07]]
```

5 Экспериментальное определение числа обусловленности матрицы

5.1 Часть листинга для нахождения числа обусловленности

```
X = solve(A, B) # find X in AX=B

# find absolute maximal element in vector B
max_B_id = np.argmax(np.abs(B))

# create delta B vector with one non-zero element, which
# equals to 1% of the absolute maximal element in vector B.
deltaB = np.zeros(B.shape)
deltaB[max_B_id] = B[max_B_id] * 0.01

X2 = solve(A, B + deltaB) # find deltaX in A * deltaX = B + deltaB

# Calculate and print condition number
print("cond(A) is greater or equals to: ")
print(norm(X2 - X) / norm(X) * norm(B) / norm(deltaB))
```

5.2 Полученные результаты

При нахождении числа обусловленности экспериментальным методом, для расчётов неизвестных ($\boldsymbol{X}$) использовался метод исключения Гаусса, для получения более высокой точности. А также, из-за того, что итерационным методом некоторые СЛАУ не возможно решить.

5.2.0.1 Пример (1)

```
cond(A) is greater or equals to:
1.24391723884
```

5.2.0.2 Пример (2)

```
cond(A) is greater or equals to:
1.01289623438
```

5.2.0.3 Пример (3)

```
cond(A) is greater or equals to:
0.878771848707
```

5.2.0.4 Пример (4)

```
cond(A) is greater or equals to:
1.00300382299
```

5.2.0.5 Пример (5)

```
cond(A) is greater or equals to:
227239.749213
```

6 Дополнительные испытания

В ходе сдачи лабораторной работы было необходимо провести дополнительные испытания на следующей матрице:

$$\begin{cases} x_{11} + 2x_{12} + 3x_{13} = 10 \\ x_{21} + 3x_{22} - 2x_{23} = 7 \\ 2x_{31} - x_{32} + x_{33} = 5 \end{cases} \tag{6}$$

6.0.0.1 Решение СЛАУ с помощью метода исключения Гаусса

```
                3                2                1
```

6.0.0.2 Решение СЛАУ с помощью метода Гаусса-Зейделя

```
I = 0
X =                7              1.5             -7.5
ACC = 7.5
I = 1
X =            -12.5             22.5             52.5
ACC = 60
does not converge
```

Как можно видеть из результатов, данный алгоритм не сходится. Это обусловлено тем, что для данной СЛАУ не выполняется необходимое условие сходимости.

При преобразовании СЛАУ, путем перестановки третьего и первого уравнения местами, начинает выполнятся условие сходимости:

$$\begin{cases} 2x_{31} - x_{32} + x_{33} = 5 \\ x_{11} + 2x_{12} + 3x_{13} = 10 \\ x_{21} + 3x_{22} - 2x_{23} = 7 \end{cases} \tag{7}$$

```
I = 0
X =            2.5          1.5          1.5
ACC = 2.5
I = 1
X =            2.5          2.5     0.833333
ACC = 1
I = 2
X =        3.33333      1.77778      1.03704
ACC = 0.833333
I = 3
X =        2.87037       2.0679     0.997942
ACC = 0.462963
I = 4
X =        3.03498      1.98697     0.997028
ACC = 0.164609
I = 5
X =        2.99497       1.9997      1.00188
ACC = 0.0400091
I = 6
X =        2.99891      2.00162     0.999286
ACC = 0.00393741
I = 7
X =        3.00117      1.99914      1.00019
ACC = 0.00248193
I = 8
X =        2.99947       2.0003     0.999975
ACC = 0.00169194
I = 9
X =        3.00016      1.99993     0.999993
ACC = 0.000689077
       3.00016      1.99993     0.999993
```

7 Выводы

В ходе выполнения данной лабораторной работы были реализованы два алгоритма нахождения решения СЛАУ на языке C++:

- Прямой метод: «метод исключения Гаусса»;
- Итерационный метод: «метод Гаусса-Зейделя».

А также были вычислены числа обусловленности для заданных примеров экспериментальным путём.

При сравнении двух реализованных методов можно заметить, что метод Гаусса-Зейделя не предназначен для решения всевозможных СЛАУ. При проверки работы алгоритмов на предоставленных примерах, в 1-ом и 2-ом примере метод Гаусса-Зейделя не сходиться. Примеры 3 и 4 имели довольно схожие результаты. 5-ый пример при низкой точности ($\varepsilon \approx 10^{-6}$), результаты, полученные методом Гаусса-Зейделя довольно сильно расходиться с методом исключения Гаусса.

Разница времени выполнения обоих методов для данных (малых) СЛАУ была довольно незначительной, поэтому, для сравнения времени выполнения, было принято решение создать СЛАУ размером 1500×1500, и произвести замеры на ней.

Каждый алгоритм был запущен 10 раз на одной и той-же матрице 1500×1500, в результате получено:

Метод	Среднее время выполнения (c)	Точность
Метод исключения Гаусса	5.357956087	$\pm 0.33\%$
Метод Гаусса-Зейделя	0.655645297	$\pm 0.73\%$

Полученные результаты были ожидаемы, т.к. метод исключения Гаусса имеет алгоритмическую сложность $O(n^3)$, тогда как метод Гаусса-Зейделя: $O(mn^2)$, где m – количество итераций, которое необходимо для достижения требуемой точности.

$$O(n^3) > O(mn^2) \qquad \text{если } n \gg m$$

Таким образом при не больших СЛАУ (когда $n \approx m$) метод исключения Гаусса не медленнее метода Гаусса-Зейделя. Однако при увеличении размера СЛАУ, когда $n \gg m$, первый метод начинает работать значительно медленнее второго.

В ходе вычисления числа обусловленности Матриц была использована Евклидова норма. За исключением 5-ого примера, все числа обусловленности получились до 10, что означает хорошую обусловленность. Число обусловленности матрицы из 5-ого уравнения составила ≈ 227240, что явно свидетельствует о плохой обусловленности, так как значение выше 1000. Скорее всего, данный факт и является причиной расхождения результатов метода Гаусса-Зейделя и метода исключений Гаусса при низкой точности.