Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №1 «Решение системы линейных алгебраических уравнений СЛАУ»

по дисциплине «**Вычислительная математика**»

Автор: Антоневич Глеб Владимирович

Факультет: ФПИиКТ

Группа: P3211

Преподаватель: Малышева Татьяна Алексеевна

Санкт-Петербург 2021

Вариант 1, Метод Гаусса

Цель работы: реализовать программу для решения системы линейных уравнений методом Гаусса.

Описание метода: Суть метода заключается в преобразования расширенной СЛАУ к треугольному виду и последующему нахождению всех неизвестных. Если матрица квадратная и она имеет определитель, не равный нулю, то мы имеем единственное решение. Далее мы находим все неизвестные начиная с последней строки. Каждая неизвестная выражается через предыдущие, а последняя известна сразу.

Расчетные формулы:

Прямой ход:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}, \\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + \ldots + a_{3n}^{(1)}x_n &= b_3^{(1)} \\ a_{n2}^{(1)}x_2 + a_{n3}^{(1)}x_3 + \ldots + a_{nn}^{(1)}x_n &= b_n^{(1)} \end{aligned}$$

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j}\,, i,j = 2,3\ldots n$$

$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}}b_1\,, i = 2,3\ldots n$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\ a_{n3}^{(2)}x_3 + \ldots + a_{nn}^{(2)}x_n &= b_n^{(2)} \end{aligned}$$

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}a_{2j}^{(1)}\,, \quad i,j = 3,4\ldots n \qquad b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}b_2^{(1)}\,, \quad i = 3,4\ldots n$$

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1, \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)}, \\ a_{33}^{(2)}x_3 + \ldots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\ a_{nn}^{(n-1)}x_n &= b_n^{(n-1)} \end{aligned}$$

Обратный ход:

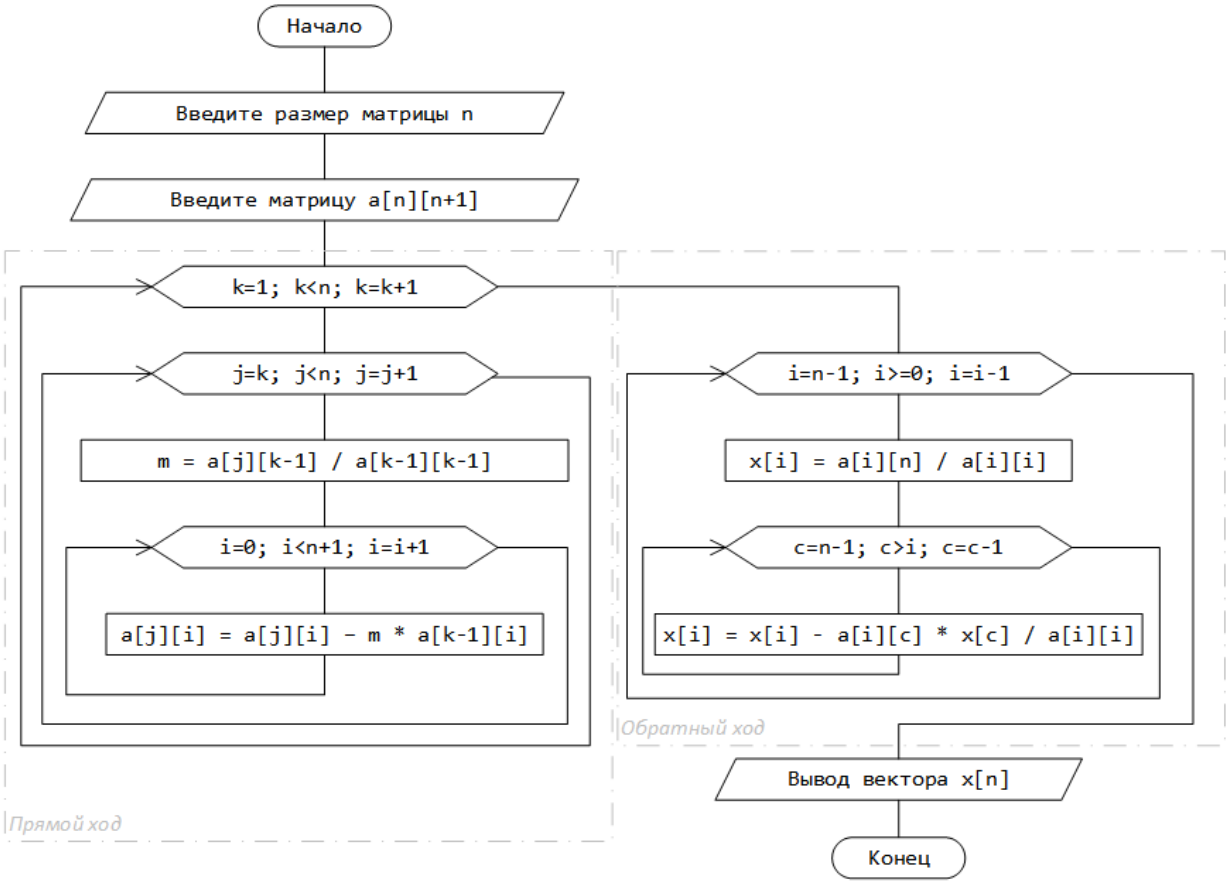
$$x_n = b_n^{(n-1)} \Big/ a_{nn}^{(n-`1)}$$

……

$$x_2 = \frac{1}{a_{22}^{(1)}}(b_2^{(1)} - a_{23}^{(1)}x_3 - \cdots - a_{2n}^{(1)}x_n)$$

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)$$

Блок схема численного метода:



Листинг программы:

Основные функции

```python
def get_solutions(matrix: List[List[float]], precision: Optional[int]) -> List[float]:
    a: List[List[float]] = copy.deepcopy(matrix)
    n: int = len(a)
    print_matrix_and_precision_and_iteration(a, precision)
    for k in range(1, n):
        for j in range(k, n):
            __check_zero_division(a, k - 1)
            m: float = __round(a[j][k - 1] / a[k - 1][k - 1], precision)
            for i in range(n + 1):
                a[j][i] = __round(a[j][i] - m * a[k - 1][i], precision)
        print_matrix_and_precision_and_iteration(a, precision, iteration=k)

    x: List[float] = [0 for _ in range(n)]
    for i in range(n - 1, -1, -1):
        __check_zero_division(a, i)
        x[i] = __round(a[i][-1] / a[i][i], precision)
        for c in range(n - 1, i, -1):
            x[i] = __round(x[i] - (a[i][c] * x[c] / a[i][i]), precision)
    return x
```

```python
def get_residuals(matrix: List[List[float]], solution: List[float], precision: Optional[int]) -> List[float]:
    matrix_right: List[float] = [row[-1] for row in matrix]
    temp: List[float] = [0 for _ in range(len(matrix))]

    residuals: List[float] = [0 for _ in range(len(matrix))]
    for i in range(len(matrix[0]) - 1):
        temp[i] = 0
        for j in range(len(matrix[0]) - 1):
            temp[i] = __round(temp[i] + solution[j] * matrix[i][j], precision)
        residuals[i] = __round(temp[i] - matrix_right[i], precision)

    return residuals
```

```python
def determinant_recursive(matrix: List[List[float]], determinant: float = 0) -> float:
    indices: List[int] = list(range(len(matrix)))

    if len(matrix) == 2 and len(matrix[0]) == 2:
        return matrix[0][0] * matrix[1][1] - matrix[1][0] * matrix[0][1]

    for focus_column in indices:
        submatrix: List[List[float]] = copy.deepcopy(matrix)
        submatrix = submatrix[1:]

        for i in range(len(submatrix)):
            submatrix[i] = submatrix[i][0:focus_column] + submatrix[i][focus_column + 1:]

        sign: int = (-1) ** (focus_column % 2)

        subdeterminant: float = determinant_recursive(submatrix)
        determinant += sign * matrix[0][focus_column] * subdeterminant

    return determinant
```

Вспомогательные:

```python
EPSILON = 1e-08
decimal.getcontext().rounding = decimal.ROUND_HALF_UP  # round(0.5) → 1.0


def __is_close(first: float, second: float) -> bool:
    return isclose(first, second, rel_tol=EPSILON, abs_tol=EPSILON)


def __round(number: float, precision: Optional[int] = None) -> float:
    if precision is not None:
        number_decimal: decimal.Decimal = decimal.Decimal(str(number))
        number_rounded = round(number_decimal, precision)
        return float(number_rounded)
    return number


def __check_zero_division(matrix: List[List[float]], index: int) -> None:
    if __is_close(matrix[index][index], 0):
        if all(__is_close(matrix[index][j], 0) for j in range(len(matrix[index]))):
            raise Exception('Infinite many solutions')
        else:
            raise Exception('No solutions')
```

Примеры:

```
Gauss method

Enter action:
0 - exit
1 - enter from random
2 - enter from file
3 - enter from console
1
Enter matrix size (default = 5): 7
Enter values type (int/float) (default = int):
Enter solutions range (default = -10 10): -42 42
Enter coefficient range (default = -100 100):

matrix:
-34 -19  59 25  -4  34  24  4218
 33  78 -12 65 -56 -18  89   153
 35  61  70 34  10 -79  50  -621
 63 -97  97 91 -80  26 -73 -4815
-70 -56 -96 42  95 -58 -41   129
 95 -45  68 24  25 -44 -58 -5031
 10  79  54 66  85  -7 -82   711

iteration: 1
matrix:
-34         -19          59         25         -4   34         24         4218
  0   59.558824   45.264706  89.264706 -59.882353   15 112.294118  4246.941176
  0   41.441176  130.735294  59.735294   5.882353  -44  74.705882  3721.058824
  0 -132.205882  206.323529 137.323529 -87.411765   89 -28.529412  3000.705882
  0  -16.882353 -217.470588  -9.470588 103.235294 -128 -90.411765 -8555.117647
  0  -98.088235  232.852941  93.852941  13.823529   51   9.058824  6754.588235
  0   73.411765   71.352941  73.352941  83.823529    3 -74.941176  1951.588235

iteration: 2
matrix:
-34       -19        59         25          -4          34          24         4218
  0 59.558824 45.264706  89.264706  -59.882353          15  112.294118  4246.941176
  0         0     99.24  -2.375309   47.548642  -54.437037   -3.428642   766.026667
  0         0     306.8 335.469136 -220.335802  122.296296  220.735802 12427.866667
  0         0   -204.64  15.832099   86.261235 -123.748148  -58.581235 -7351.293333
  0         0     307.4 240.864198  -84.797531   75.703704  193.997531 13748.933333
  0         0     15.56 -36.674074  157.634074  -15.488889 -213.354074     -3283.16

iteration: 3
matrix:
-34       -19        59         25          -4          34          24         4218
  0 59.558824 45.264706  89.264706  -59.882353          15  112.294118  4246.941176
  0         0     99.24  -2.375309   47.548642  -54.437037   -3.428642   766.026667
  0         0         0 342.812391 -367.332209  290.588144  231.335433 10059.698777
  0         0         0  10.934042  184.309946 -236.001224   -65.65134 -5771.691388
  0         0         0 248.221814 -232.081414  244.324675  204.617891 11376.134086
  0         0         0 -36.301646  150.178846   -6.953618 -212.816492 -3403.266559
```

```
iteration: 4
matrix:
-34       -19        59         25          -4          34          24         4218
  0 59.558824 45.264706  89.264706  -59.882353          15  112.294118  4246.941176
  0         0     99.24  -2.375309   47.548642  -54.437037   -3.428642   766.026667
  0         0         0 342.812391 -367.332209  290.588144  231.335433 10059.698777
  0         0         0          0   196.02605 -245.269567  -73.029811 -6092.546675
  0         0         0          0   33.894582   33.917122   37.113733  4092.159694
  0         0         0          0  111.280708   23.817813 -188.319545 -2338.008625

iteration: 5
matrix:
-34       -19        59         25          -4          34          24         4218
  0 59.558824 45.264706  89.264706  -59.882353          15  112.294118  4246.941176
  0         0     99.24  -2.375309   47.548642  -54.437037   -3.428642   766.026667
  0         0         0 342.812391 -367.332209  290.588144  231.335433 10059.698777
  0         0         0          0   196.02605 -245.269567  -73.029811 -6092.546675
  0         0         0          0           0    76.32633   49.741213  5145.613171
  0         0         0         -0          -0  163.053241 -146.861743  1120.628152

iteration: 6
matrix:
-34       -19        59         25          -4          34          24         4218
  0 59.558824 45.264706  89.264706  -59.882353          15  112.294118  4246.941176
  0         0     99.24  -2.375309   47.548642  -54.437037   -3.428642   766.026667
  0         0         0 342.812391 -367.332209  290.588144  231.335433 10059.698777
  0         0         0          0   196.02605 -245.269567  -73.029811 -6092.546675
  0         0         0          0           0    76.32633   49.741213  5145.613171
  0         0         0         -0          -0           0 -253.122138 -9871.763382

determinant: 260907519777876

solutions:
x[1] = -30
x[2] = 3
x[3] = 15
x[4] = 6
x[5] = 36
x[6] = 42
x[7] = 39

residuals:
r[1] = 0
r[2] = 0
r[3] = 0
r[4] = -0
r[5] = -0
r[6] = -0
r[7] = -0
```

```
Enter action:
0 - exit
1 - enter from random
2 - enter from file
3 - enter from console
2
Enter file name: 3x4
Enter optional floating point precision and then enter matrix:
3
6.322892 -61.587136  12.094795 124.943374
-67.690001  32.348998 -67.547541 339.481179
-89.397588 -10.989369  61.474078 140.767714

floating point precision: 3
matrix:
  6.323 -61.587  12.095 124.943
 -67.69  32.349 -67.548 339.481
-89.398 -10.989  61.474 140.768

iteration: 1
floating point precision: 3
matrix:
 6.323  -61.587  12.095  124.943
-0.002  -626.94  61.929 1676.996
 0.003 -881.768 232.485 1907.337

iteration: 2
floating point precision: 3
matrix:
 6.323 -61.587  12.095  124.943
-0.002 -626.94  61.929 1676.996
 0.006   -0.29 145.413 -450.519

determinant: -576321.635

solutions:
x[1] = -3.349
x[2] = -2.981
x[3] = -3.098

residuals:
r[1] = 0.002
r[2] = 0.045
r[3] = 0.938
```

```
Enter action:
0 - exit
1 - enter from random
2 - enter from file
3 - enter from console
3
Enter optional floating point precision and then enter matrix:
1 2 -3 5 1
1 3 -13 22 -1
3 5 1 -2 5
2 3 4 -7 4
determinant is zero, so either there are no solutions, or there are infinitely many solutions

matrix:
1 2  -3  5  1
1 3 -13 22 -1
3 5   1 -2  5
2 3   4 -7  4

iteration: 1
matrix:
1  2  -3   5  1
0  1 -10  17 -2
0 -1  10 -17  2
0 -1  10 -17  2

iteration: 2
matrix:
1 2  -3  5  1
0 1 -10 17 -2
0 0   0  0  0
0 0   0  0  0

Infinite many solutions
```

```
Enter action:
0 - exit
1 - enter from random
2 - enter from file
3 - enter from console
3
Enter optional floating point precision and then enter matrix:
1 -2 3 -4 2
3 3 -5 1 -3
-2 1 2 -3 5
3 0 3 -10 8
determinant is zero, so either there are no solutions, or there are infinitely many solutions

matrix:
 1 -2  3  -4  2
 3  3 -5   1 -3
-2  1  2  -3  5
 3  0  3 -10  8

iteration: 1
matrix:
1 -2   3  -4  2
0  9 -14  13 -9
0 -3   8 -11  9
0  6  -6   2  2

iteration: 2
matrix:
1 -2        3        -4  2
0  9      -14        13 -9
0  0 3.333333 -6.666667  6
0  0 3.333333 -6.666667  8

iteration: 3
matrix:
1 -2        3        -4  2
0  9      -14        13 -9
0  0 3.333333 -6.666667  6
0  0        0        -0  2

No solutions
```

Вывод: в ходе выполнения лабораторной работы я познакомился с различными численными методами решения СЛАУ и реализовал один из методов на языке Python. Метод подходит для сравнительно небольших матриц, на матрицах большого размера данный метод не является самым эффективным (вся матрица хранится в памяти, используется результат предыдущих вычислений, что при округлении может привести к погрешностям в процессе решения). В 1969 году Штрассен доказал, что большие матрицы можно перемножить за время O(n^(log2 7)) = O(n^2.81). Отсюда вытекает, что обращение матриц и решение СЛАУ можно осуществлять алгоритмами асимптотически более быстрыми по порядку, чем метод Гаусса. Таким образом, для больших СЛАУ метод Гаусса не оптимален по скорости O(n^3).