МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики (ИЦТЭФ)
Кафедра вычислительной техники и электроники (ВТиЭ)

Разработка и отладка параллельной MPI-программы для метода Гаусса для решения СЛАУ

Отчет по лабораторной работе №3

Выполнил: студент гр. 5.306М
______________ А. В. Лаптев
Проверил: ст. преп. кафедры ВТиЭ
______________ И. А. Шмаков
«____» ____________ 2024г.

Барнаул, 2024г.

СОДЕРЖАНИЕ

Краткое описание метода Гаусса

Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа:

1. прямой ход решения;
2. обратный ход решения.

На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. Для этого среди элементов первого столбца матрицы выбирают ненулевой, перемещают содержащую его строку в крайнее верхнее положение, делая эту строку первой. Далее ненулевые элементы первого столбца всех нижележащих строк обнуляются путём вычитания из каждой строки первой строки, домноженной на отношение первого элемента этих строк к первому элементу первой строки. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают, пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.

На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

Блок-схема алгоритма

Блок-схема алгоритма состоит из нескольких частей. На рис. 1 представлена блок-схема для выбора режима работы программы.

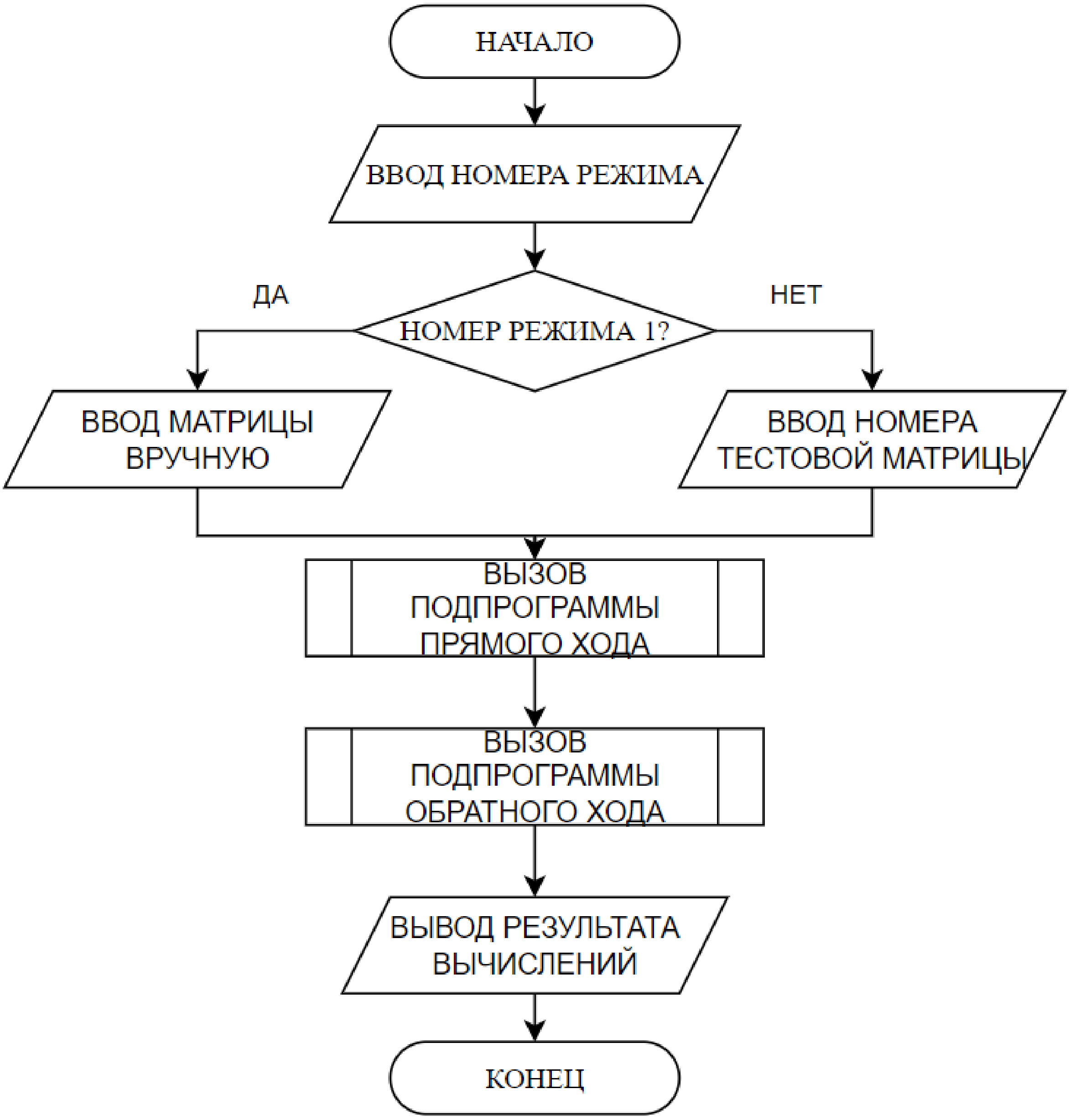

Рис. 1 Выбор режима работы программы.

На блок-схемах на рис. 2 и рис. 3 представленных ниже показано все, что касается первого этапа решения уравнения, а именно прямой проход.

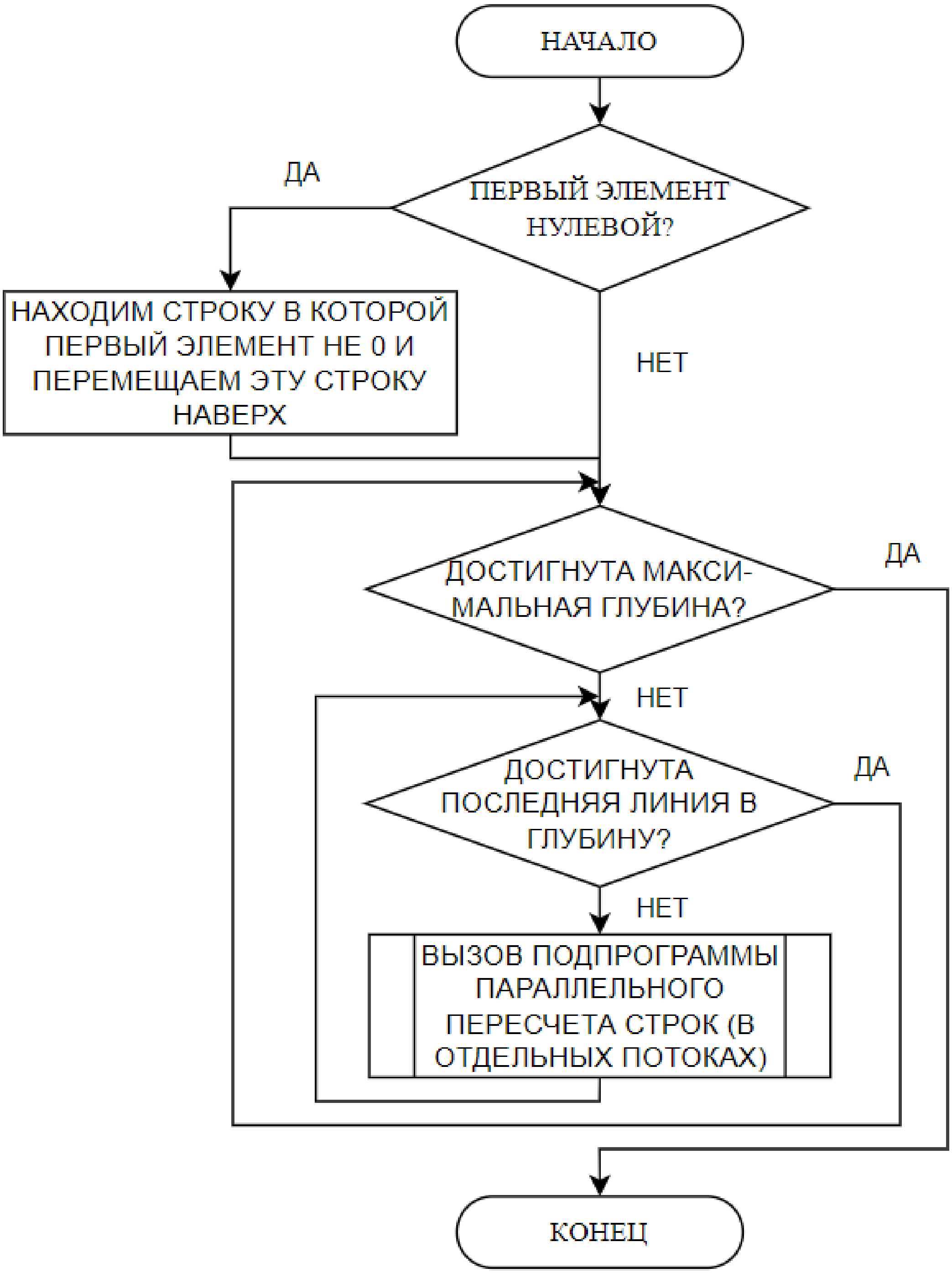

Рис. 2 Блок-схема для этапа прямого хода решения СЛАУ.

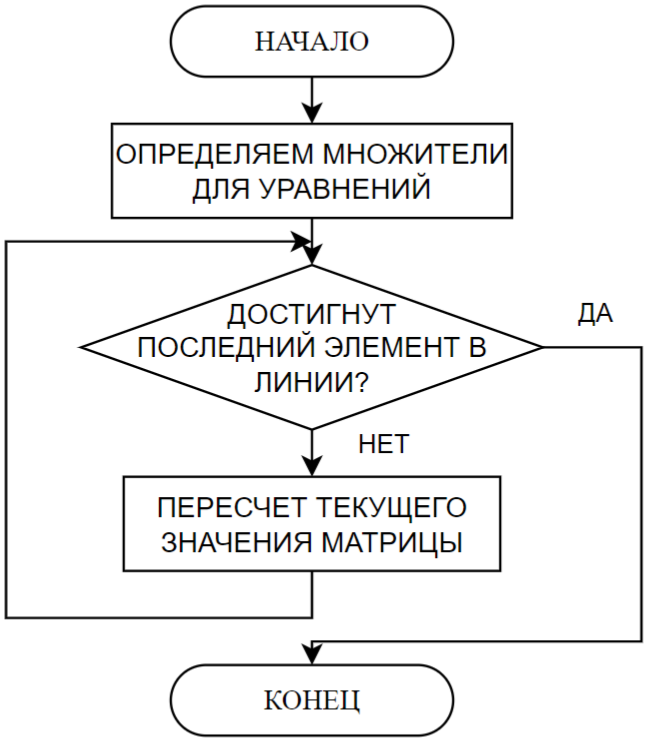

Рис. 3 Подпрограмма для параллельного суммирования i-го уравнения с первым уравнением.

На рис. 4 и рис. 5 представлены блок-схемы для иллюстрации второго этапа решения СЛАУ, а именно обратного хода решения матрицы.

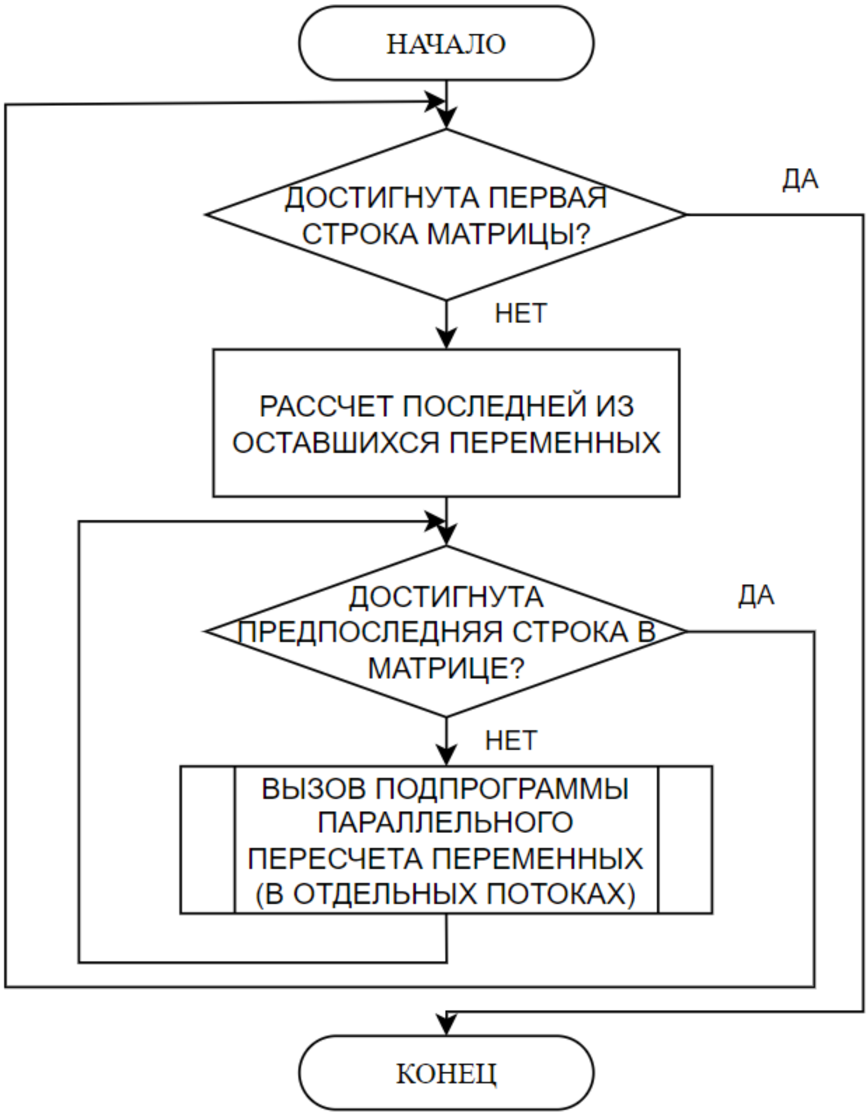

Рис. 4 Блок-схема для этапа обратного хода решения СЛАУ.

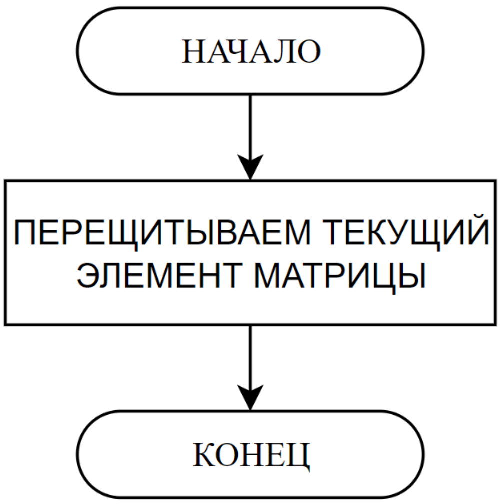

Рис. 5 Подпрограмма для параллельного обновления значений переменных всех столбцов матрицы.

Тестирование работоспособности программы

Для тестирования работоспособности программы предусмотрен отдельный режим, где можно выбрать одну из пяти СЛАУ, для которой можно сравнить результат работы программы с правильным ответом.

В качестве примера для подтверждения работоспособности взята следующая СЛАУ:

$$\begin{cases} -4x_1 + 6x_2 - 10x_3 - 2x_4 - 2x_5 = -72, \\ -6x_1 + 4x_2 - 4x_3 - 8x_4 - 9x_5 = -67, \\ 7x_1 - 5x_2 - 9x_3 - 7x_4 - 9x_5 = -51, \\ x_1 - 6x_2 + 5x_4 - 2x_5 = -44, \\ 8x_1 + 4x_2 + 9x_3 + x_4 - 4x_5 = 22. \end{cases}$$

На первом этапе нужно привести матрицу к ступенчатому виду. Поскольку первый элемент первой строки не нулевой, то никаких перестановок не требуется. Поэтому нужно нужно последовательно суммировать все уравнения, начиная со второго, с первым таким образом, чтобы получить все нули в первом столбце (для всех уравнений, кроме первого), затем для полученной системы складывать все уравнения, начиная с третьего, со вторым, чтобы получить нули во всех строках ниже второй и т.д.

Результат, полученный после выполнения первого этапа:

$$\begin{cases} -2x_1 + 3x_2 - 5x_3 - x_4 - x_5 = -36, \\ 5x_2 - 11x_3 + 5x_4 + 6x_5 = -41, \\ 144x_3 + 160x_4 + 191x_5 = 1319, \\ 1640x_4 - 1393x_5 = -1057, \\ x_5 = 9. \end{cases}$$

На втором этапе нужно раскрутить полученную систему в обратную сторону. Уже известно значение переменной x_5, а значит его можно последовательно подставлять в уравнения выше и находить остальные неизвестные коэффициенты.

Правильным ответом в ходе решения этой СЛАУ являются следующие значения: $x_1 = 3, x_2 = -1, x_3 = 5, x_4 = -7, x_5 = 9$.

Ниже представлен скриншот работы программы для данной системы уравнений. На выход программы поступает матрица, которая составлена на основе приведенной СЛАУ; список с полученными значениями переменных; время выполнения скрипта.

```
Введите номер режима (1 - ввод вручную, 2 - тестовые матрицы):  2
Введите номер тестовой матрицы (от 1 до 5):     4
[[-4, 6, -10, -2, -2, -72], [-6, 4, -4, -8, -9, -67], [7, -5, -9, -7, -9, -51], [1, -6, 0, 5, -2, -44], [8, 4, 9, 1, -4, 22]]
[3.0, -1.0, 5.0, -7.0, 9.0]
0.04630446434020996
```

Рис. 6 Результат работы программы для тестовой СЛАУ.

Вывод

В ходе выполнения работы было осуществлено знакомство с методом Гаусса для решения СЛАУ и создана и протестирована его программная реализация.

ПРИЛОЖЕНИЕ

Листинг 1 Исходный код для решения СЛАУ методом Гаусса.

```
import threading
import time


def example_matrixes(num_example):
    # ПРимеры матриц, для которых точно есть известное решение
    if num_example == 1:
        # Матрица 2x3
        matrix = [[3, 2, 16],
                  [2, -1, 6]]
        line = 2
        row = 3
    elif num_example == 2:
        # Матрица 3x4
        matrix = [[1, 2, 3, 1],
                  [2, -1, 2, 6],
                  [1, 1, 5, -1]]
        line = 3
        row = 4
    elif num_example == 3:
        # Матрица 4x5
        matrix = [[2, 5, 4, 1, 20],
                  [1, 3, 2, 1, 11],
                  [2, 10, 9, 7, 40],
                  [3, 8, 9, 2, 37]]
        line = 4
        row = 5
    elif num_example == 4:
        # Матрица 5x6
        matrix = [[-4, 6, -10, -2, -2, -72],
                  [-6, 4, -4, -8, -9, -67],
```

```python
                  [7, -5, -9, -7, -9, -51],
                  [1, -6, 0, 5, -2, -44],
                  [8, 4, 9, 1, -4, 22]]
        line = 5
        row = 6
    else:
        # Матрица 6x7
        matrix = [[4, -9, 6, 1, 4, -6, -52],
                  [-4, 1, 4, -1, 9, 1, 83],
                  [9, -6, -1, 8, -8, -4, -183],
                  [4, -3, 7, -5, -7, 3, 21],
                  [6, 2, -2, 6, -7, -3, -123],
                  [6, -4, 7, 9, 8, -6, -93]]
        line = 6
        row = 7
    return matrix, line, row


def ladder(deep_matrix, i):
    # Функция для распараллеливания суммирования уравнений ниже i-го уравнения с i-м
    first_factor = matrix[i][deep_matrix]
    second_factor = matrix[deep_matrix][deep_matrix]
    for j in range(deep_matrix, row):
        matrix[i][j] = matrix[i][j] * second_factor + \
            matrix[deep_matrix][j] * (-first_factor)


def straight_stroke(matrix, line):
    # Прямой ход в решении СЛАУ методом Гаусса (1 этап)
    if matrix[0][0] == 0:
        for i in range(1, line):
            if matrix[i][0] > 0:
                matrix.insert(0, matrix[i])
                matrix.pop(i + 1)
```

```
                break

    for deep_matrix in range(line - 1):
        for i in range(deep_matrix + 1, line):
            matrix_member_str = threading.Thread(
                target=ladder, args=(deep_matrix, i,))
            matrix_member_str.start()
        matrix_member_str.join()
    return matrix


def ladder_reverse(x, count_lines, i):
    # Функция для распараллеливания подстановки извесных коэффициентов
    matrix[count_lines][i] = matrix[count_lines][i + 1] - \
        (matrix[count_lines][i] * x[i])


def reverse_stroke(matrix, line):
    # ОБратный ход в решении СЛАУ методом Гаусса (2 этап)
    x = [0] * line
    for i in range(line - 1, -1, -1):
        x[i] = matrix[i][i + 1] / matrix[i][i]
        for count_lines in range(i - 1, -1, -1):
            matrix_member_rev = threading.Thread(
                target=ladder_reverse, args=(x, count_lines, i,))
            matrix_member_rev.start()
        matrix_member_rev.join()
    return x


if __name__ == "__main__":
    mode = 0
    while mode < 1 or mode > 2:
        mode = int(
```

```
            input("Введите номер режима (1 - ввод вручную, 2 - тестовые матрицы):\t"))

    matrix = []
    if mode == 1:
        # Задаем матрицу вручную
        line = int(input("Введите количество строк:\t"))
        row = int(input("Введите количество столбцов:\t"))
        for i in range(line):
            matrix_line = []
            for j in range(row):
                element = int(input(f"Введите элемент ({i};{j}):\t"))
                matrix_line.append(element)
            matrix.append(matrix_line)
    elif mode == 2:
        num_example = int(input("Введите номер тестовой матрицы (от 1 до 5):\t"))
        # Тестовые наборы матриц
        matrix = example_matrixes(num_example)[0]
        line = example_matrixes(num_example)[1]
        row = example_matrixes(num_example)[2]
    print(matrix)
    start_time = time.time()
    straight_stroke(matrix, line)
    x = reverse_stroke(matrix, line)
    end_time = time.time()
    print(x)
    print(end_time - start_time)
```