Метод Гаусса с выбором главного элемента по строке

Библиотеки:

In [1]:

```
import numpy as np
from scipy import linalg as LA
import math
```

Для начала напишем ряд вспомогательных функций, которые пригодятся потом в основной:

In [2]:

```
def makeI(n):
    # возвращает единичную матрицу размера n*n

    I = []
    for i in range(n):
        k = np.zeros(n)
        k[i] = 1
        I.append(k)
    return np.array(I)


def sq_matrix_product(A, B):
    # произведение квадратных матриц A@B

    n = len(A)
    C  = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
    return C


def matrix_and_column_product(A, b):
    # произведение квадратной матрицы и столбца

    n = b.shape[0]
    c = np.zeros(n)

    for i in range(n):
        for j in range(n):
            c[i] = c[i] +  A[i][j] * b[j]

    return c


def find_Q(A):
    # находит матрицу Q - перестановок, соответствующих выбору главного элемента по строке

    n = len(A)
    Q = makeI(n)
    for i in range(n - 1):
        max_ind = i
        max_val = abs(A[i, i])
        for j in range(i, n):
            if (abs(A[i, j]) > max_val):
                max_val = abs(A[i, j])
                max_ind = j

        if max_ind is not i:
            for k in range(n):
                temp = Q[k, max_ind]
                Q[k, max_ind] = Q[k, i]
                Q[k, i] = temp

    return Q
```

```python
def findLU(A):
    # вычисляет LU-разложение для матрицы A

    U = np.copy(A)
    n = len(A)
    # единичная матрица
    L = makeI(n)

    for i in range(0, n-1):
        for j in range(i+1, n):
            L[j, i] = U[j, i] / U[i, i]
            U[j, i:n] = U[j, i:n] - L[j, i] * U[i, i:n]

    L = np.array(L)
    U = np.array(U)
    return L, U
```

И вот основная функция, которая решает систему нужным методом:

In [3]:

```
def find_gauss(A, b):
    # принимает на вход матрицу A и правую часть b

    n = len(A)
    A = np.array(A)
    b = np.array(b)

    # вычисление матрицы перестановки
    Q = find_Q(A)

    # матричное произведение
    AQ = sq_matrix_product(A, Q)

    # ищем LU-разложение: AQ=LU
    L, U = findLU(AQ)

    # прямой проход
    y = np.zeros(n)
    for i in range(n):
        y[i] = b[i]
        for j in range(i):
            y[i] -= L[i, j] * y[j]

    # обратный проход
    x = np.zeros(n)
    for i in reversed(range(n)):
        x[i] = y[i]

        if (i < n):
            for j in range(i + 1, n):
                x[i] -= x[j] * U[i, j]
        x[i] /= U[i,i]

    # см. пояснение в клетке ниже
    x = matrix_and_column_product(Q, x)

    # возвращаем матрицы и вектор решения
    return L, U, Q, x
```

Пояснение к строке: x = matrix_and_column_product(Q, x)

Мы нашли x такой, что $AQx = b$

Но нам нужен $\widehat{x}$, такой что $A\widehat{x} = b$.

Таким образом: $A\widehat{x} = AQX \Rightarrow \widehat{x} = Qx$, что мы и написали в предпоследней строчке функции.

Проверка решения

Рассмотрим теперь случайные матрицу A и правую часть b.

In [4]:

```
# размер A
N = 8

A = np.random.randint(-1000, 1000, (N, N))
b = np.random.randint(-1000, 1000, N)
```

In [5]:

```
# решение при помощи библиотечной функции
x_real = np.linalg.solve(A, b)

# наше решение
x_our = find_gauss(A,b)[3]

l1_norm = np.linalg.norm(x_our - x_real, 1)

print("Решение: ", x_our)
print("Норма разницы решений:", l1_norm)
```

```
Решение:  [ 7.15059238 -4.91212161 -1.91284704 -2.81517714  6.38611862 -2.98
437887
 -5.3983932  -1.94039785]
Норма разницы решений: 6.639133687258436e-14
```

В принципе, функцию для вычисления нормы также можно расписать. Рассмотрим, например, такую:

In [6]:

```
def find_norm(x, y):
    n = len(x)
    res = 0

    for i in range(n):
        res += (x[i] - y[i]) ** 2

    res = np.sqrt(res)

    return res
```

In [7]:

```
print("Посчитанная норма разницы решений:", find_norm(x_our, x_real))
```

```
Посчитанная норма разницы решений: 2.6720187556070636e-14
```

В любм случае, можно заметить, что норма разницы между двумя решениями довольно маленькая.