Задание 3.1.

Реализовать решение СЛАУ с помощью LU разложения,
котороереализовано в виде двух функций, одна из которых возвращает две матрицы – **L** и **U**, не модифицируя **A**, а вторая функция решает систему,реализовано в виде двух функций, одна из которых возвращает две матрицы – L и U, не модифицируя A, а вторая функция решает систему, и с помощью LU разложения по схеме частичного выбора, которое модифицирует исходную матрицу A модифицирует исходную матрицу A. Решить систему небольшой размерности с возмущенной матрицей обоими методами, оценить погрешность и сравнить с теоретической оценкой. Проанализировать поведение методов с ростом числа уравнений.

$$A_{i,j} = \sin^{20-j}(i+1)$$

N = 21; K = 1

1. Реализовать метод решения СЛАУ с помощью LU разложения в виде, двух функций, одна из которых возвращает две матрицы – L и U, **не модифицируя A**, а вторая функция решает систему. Убедиться в его работоспособности.

```
#создание матриц L и U
def LU (A):
    n = A.shape[0]
    LU = np.matrix(np.zeros((n, n)))
    for t in range(n):
        for j in range(t, n):
            LU[t, j] = A[t, j] - LU[t, :t] * LU[:t, j]
        for i in range(t + 1, n):
            LU[i, t] = (A[i, t] - LU[i, :t] * LU[:t, t]) / LU[t, t]

    L = LU.copy()
    U = LU.copy()
    for i in range(n):
        L[i, i] = 1
        L[i, i + 1 :] = 0
    for j in range(1, n):
        U[j, :j] = 0

    return L, U
```

```
#решение системы
def solve_LU (L, U, b):
    n = L.shape[0]
    y = np.zeros((n, 1))
    for i in range(n):
        y[i] = b[i] - L[i, :i] * y[:i]

    x = np.zeros((n, 1))
    for i in range(1, n + 1):
        x[-i] = (y[-i] - U[-i, -i:] * x[-i:]) / U[-i, -i]

    return x
```

```
#проверка
A = np.matrix([[1, 2, 9] , [0, 5, 6], [1, 8, 5]])
b = np.array([10, 11, 12])
print("x = ", linalg.solve(A, b)) #встроенная scipy функция
```

```
x =  [-1.  1.  1.]
```

```
L, U = LU(A)
print("Mtx L ", L, '\n', "Mtx U ", U)
print("x = ", solve_LU(L, U, b).reshape(3)) # наша функция
```

```
Mtx L  [[1.  0.  0. ]
 [0.  1.  0. ]
 [1.  1.2 1. ]]
 Mtx U  [[  1.    2.    9. ]
 [  0.    5.    6. ]
 [  0.    0.  -11.2]]
x =  [-1.  1.  1.]
```

2. Реализовать метод решения СЛАУ с помощью LU разложения по схеме частичного выбора, так чтобы он **модифицировал** матрицу А. Убедиться в его работоспособности.

```
#поиск максимума в столбце на k-ом шаге - перестановка строк
def modif_mtx (A, P, col):
    max_el = A[col, col]
    ind = col
    for i in range(col + 1, A.shape[0]):
        if abs(A[i, col]) > abs(max_el):
            max_el = A[i, col]
            ind = i
    if ind != col:
        A[[col, ind]] = A[[ind, col]]
        P[[col, ind]] = P[[ind, col]]
    return
```

```
#матрица A в методе с частичным выбором модифицируется
def LU_m (A, b):
    n = A.shape[0]
    P = np.eye(n)
    U = np.float64(A)
    for k in range(n - 1):
        P_ = np.eye(n)
        modif_mtx(U, P_, k)
        for i in range(k + 1, n):
            div = U[i, k] / U[k, k]
            U[i, k] = div
            if div != 0:
                for j in range(k + 1, n):
                    U[i, j] -= div * U[k, j]
        P = P_ @ P

    L = U.copy()
    for i in range(n):
        L[i, i] = 1
        L[i, i + 1 :] = 0
    for j in range(1, n):
        U[j, :j] = 0

    b = P @ b
    y = np.zeros(n)
    x = np.zeros(n)

    for i in range(n):
         y[i] = b[i] - np.sum([L[i, j] * y[j] for j in range(i)])
    for i in range(n-1, -1, -1):
        x[i] = (y[i] - np.sum([x[j] * U[i, j] for j in range(n-1, i-1, -1)])) / U[i, i]

    return P, L, U, x
```

```python
A = np.matrix([[1, 2, 6] , [4, 8, -1], [-2, 3, 5]]) # возьмём матрицу из лекций
b = np.array([1, 0, 1])
print("x = ", linalg.solve(A, b)) #встроенная scipy функция
```

```
x =  [-0.04  0.04  0.16]
```

```python
P, L, U, x = LU_m (A, b)
print("mtx P = \n", P, '\n', "mtx L = \n", L, '\n' , "mtx U = \n", U)
print("x = ", x)
```

```
mtx P = 
 [[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]] 
 mtx L = 
 [[ 1.    0.    0.  ]
 [-0.5   1.    0.  ]
 [ 0.25  0.    1.  ]] 
 mtx U = 
 [[ 4.    8.   -1.  ]
 [ 0.    7.    4.5 ]
 [ 0.    0.    6.25]]
x =  [-0.04  0.04  0.16]
```

3. Решить систему $A^{*}x = b$, размера 5x5, двумя методами. Вектор b задается как $b = Ax$, где $x_i = N$, N -- номер варианта. Матрицу A^{*}_{ij} задать как A_{ij} и к одному элементу прибавить 10^{-3}.

```python
#элемент матрицы A
def aij (i, j):
    return np.sin(i+1) ** (20 - j)
```

```python
#заполнение матрицы
def matrix_creation (n):
    return np.array([[aij(i, j) for i in range(n)] for j in range(n)])
```

```python
#матрицу и вектор b
A_def = matrix_creation(5)
#X точное
x_def = np.array([21] * 5)
#b точное
b_def = A_def @ x_def
#b возмущённое
b = b_def.copy()
b[1] += 1e-3
#A возмущенное
A = np.copy(A_def)
print(A)
```

```
[[ 3.16798348e-02  1.49320433e-01  9.81232336e-18  3.79884091e-03
   4.32201484e-01]
 [ 3.76481607e-02  1.64215172e-01  6.95317659e-17 -5.01959354e-03
  -4.50714927e-01]
 [ 4.47408899e-02  1.80595664e-01  4.92713732e-16  6.63263344e-03
   4.70021397e-01]
 [ 5.31698546e-02  1.98610112e-01  3.49145199e-15 -8.76402163e-03
  -4.90154863e-01]
 [ 6.31867950e-02  2.18421504e-01  2.47410132e-14  1.15803287e-02
   5.11150751e-01]]
```

```python
L1, U1 = LU(A)
x1 = solve_LU(L1, U1, b).reshape(5)
L2, U2, P2, x2 = LU_m(A, b)
print(x1)
print(x2)
print(linalg.solve(A, b))
```

```
[ 3.50935298e+02  1.77152419e+01 -6.12291642e+14  2.39189217e+03
 -5.01555280e+01]
[ 3.50935298e+02  1.77152419e+01 -6.12291642e+14  2.39189217e+03
 -5.01555280e+01]
[ 3.50935298e+02  1.77152419e+01 -6.12291642e+14  2.39189217e+03
 -5.01555280e+01]
```

```python
#матрицу и вектор b
A_def = matrix_creation(5)
#X точное
x_def = np.array([21] * 5)
#b точное
b_def = A_def @ x_def
#b возмущённое
b = b_def.copy()
#A возмущенное
A = np.copy(A_def)
A[0][0] += 10e-03
print(A)
```

```
[[ 4.16798348e-02  1.49320433e-01  9.81232336e-18  3.79884091e-03
   4.32201484e-01]
 [ 3.76481607e-02  1.64215172e-01  6.95317659e-17 -5.01959354e-03
  -4.50714927e-01]
 [ 4.47408899e-02  1.80595664e-01  4.92713732e-16  6.63263344e-03
   4.70021397e-01]
 [ 5.31698546e-02  1.98610112e-01  3.49145199e-15 -8.76402163e-03
  -4.90154863e-01]
 [ 6.31867950e-02  2.18421504e-01  2.47410132e-14  1.15803287e-02
   5.11150751e-01]]
```

```python
L1, U1 = LU(A)
x1 = solve_LU(L1, U1, b).reshape(5)
L2, U2, P2, x2 = LU_m(A, b)
print(x1)
print(x2)
print(linalg.solve(A, b))
```

```
[-9.52795999e+03  2.43492387e+03  5.88131062e+15 -1.81334360e+04
  2.88838660e+02]
[-9.52795999e+03  2.43492387e+03  5.88131062e+15 -1.81334360e+04
  2.88838660e+02]
[-9.52795999e+03  2.43492387e+03  5.88131062e+15 -1.81334360e+04
  2.88838660e+02]
```

Внесли погрешности по-отдельности в матрицу и в вектор.

4. Вычислить погрешность и сравнить ее с теоретической оценкой. Для вычисления обратной матрицы можно воспользоваться встроенными функциями.

```
def delta_x(x, x_def):
    return linalg.norm(x - x_def, ord = 2) / linalg.norm(x, ord = 2)
```

```
th = np.linalg.cond(A) * delta_x(b, b_def)
print(np.linalg.cond(A))
print(th)
print(delta_x(x2, x_def))
```

```
188807181953577.97
70066535423.82056
1.0000000000000342
```

```
b[0] += 10e-03
th = np.linalg.cond(A) * delta_x(b, b_def)
print(np.linalg.cond(A))
print(th)
print(delta_x(x2, x_def))
```

```
507616905909556.0
188377145146.87766
0.9999999999999964
```

Матрица A плохо обусловлена. Из этого вытекает сильная разница между точным решением и решением, которое выдают оба метода. Погрешность решения при внесении изменений в матрицу и в вектор по отдельности практически равна.

5. Задавая вектор b как $b = Ax$, где $x_i = N$, решить систему обоими методам для размера матрицы $n = 5,\ldots,15$.

```
#массивы с погрешностями
delta_x1 = []
delta_x2 = []

for i in range(5, 16):
    A_def = matrix_creation(i)
    x_def = np.array([21] * i)
    b = A_def @ x_def
    A = np.copy(A_def)
    A[0][0] += 1e-3
    L1, U1 = LU(A)
    x1 = solve_LU(L1, U1, b).reshape(i)
    L2, U2, P2, x2 = LU_m(A, b)
    delta_x1.append(delta_x(x1,x_def))
    delta_x2.append(delta_x(x2,x_def))
```

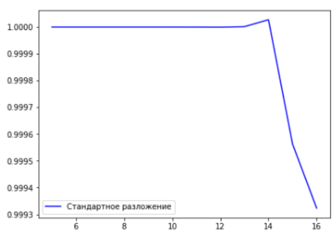
6. Построить на одном графике погрешности обоих методов как функций, зависящих от n. Прокомментировать полученный результат.

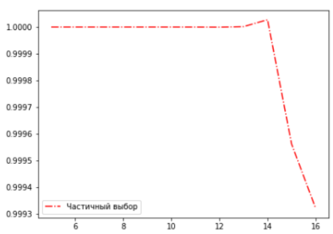
```
#массивы с погрешностями
delta_x1 = []
delta_x2 = []

for i in range(5, 17):
    A_def = matrix_creation(i)
    x_def = np.array([21] * i)
    b = A_def @ x_def
    A = np.copy(A_def)
    A[0][0] += 1e-3
    L1, U1 = LU(A)
    x1 = solve_LU(L1, U1, b).reshape(i)
    L2, U2, P2, x2 = LU_m(A, b)
    delta_x1.append(delta_x(x1,x_def))
    delta_x2.append(delta_x(x2,x_def))
```

```
from matplotlib import pyplot as plt
```

```
fig, axs = plt.subplots(1,2, figsize = (15,5))
axs[0].plot(range(5,17), delta_x1, color = 'blue', ls = 'solid',label = 'Стандартное разложение')
axs[1].plot(range(5,17), delta_x2, color = 'red', ls = 'dashdot',label = 'Частичный выбор')
axs[0].legend()
axs[1].legend()
```

```
<matplotlib.legend.Legend at 0x2101f037b50>
```





Методы 1 и 2 равносильны, так как графики погрешностей практически совпадают. Погрешности решений в районе единицы, что говорит о большой потере точности. При увеличении размерностей погрешность начинает падать.

Задание 3.2.

Дана система уравнений $Ax = b$ порядка n с разреженной матрицей A. Решить систему прямым методом.

N = 21, n = 70

На побочной диагонали элементы равны 50, в 65-ом столбце элементы равны 10.

$b_i = i^2 - 100$

1. Для указанной в индивидуальном варианте системы уравнений вывести формулы для нахождения неизвестных.

   Легко видеть, что x_{65} вычисляется сразу, так как он находится на пересечении столбца и побочной диагонали.

$$x_{65} = \frac{b_5}{50}$$

   Остальные неизвестные легко найдутся по формуле $x_i = \frac{b_{n-i+1} - 10\,x_{65}}{50}$

```
def solve_mysys (t1, t2, b):
    x = np.zeros(70)
    x[64] = b[5] / t1
    for i in range(70):
        if i != 64:
            x[i] = (b[-i - 1] - t2 * x[64]) / t1
    return x
```

2. Подготовить тестовый пример.

```
t_side_diag = 50
t_65 = 10
b = [60] * 70
b[5] = 50
solve_mysys(t_side_diag, t_65, b)
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1.])
```

   Особенность примера в том, что все элементы вектора-решения равны единице.

3. Решить систему для тестового примера и для указанной в варианте системы уравнений.

```
В [21]: b = np.array([i**2 - 100 for i in range(70)])

        print(np.around(solve_mysys(t_side_diag, t_65, b),3))
```

```
[ 9.3556e+01  9.0816e+01  8.8116e+01  8.5456e+01  8.2836e+01  8.0256e+01
  7.7716e+01  7.5216e+01  7.2756e+01  7.0336e+01  6.7956e+01  6.5616e+01
  6.3316e+01  6.1056e+01  5.8836e+01  5.6656e+01  5.4516e+01  5.2416e+01
  5.0356e+01  4.8336e+01  4.6356e+01  4.4416e+01  4.2516e+01  4.0656e+01
  3.8836e+01  3.7056e+01  3.5316e+01  3.3616e+01  3.1956e+01  3.0336e+01
  2.8756e+01  2.7216e+01  2.5716e+01  2.4256e+01  2.2836e+01  2.1456e+01
  2.0116e+01  1.8816e+01  1.7556e+01  1.6336e+01  1.5156e+01  1.4016e+01
  1.2916e+01  1.1856e+01  1.0836e+01  9.8560e+00  8.9160e+00  8.0160e+00
  7.1560e+00  6.3360e+00  5.5560e+00  4.8160e+00  4.1160e+00  3.4560e+00
  2.8360e+00  2.2560e+00  1.7160e+00  1.2160e+00  7.5600e-01  3.3600e-01
 -4.4000e-02 -3.8400e-01 -6.8400e-01 -9.4400e-01 -1.6800e+00 -1.3440e+00
 -1.4840e+00 -1.5840e+00 -1.6440e+00 -1.6640e+00]
```

Задание 3.3.

Решить задачу методом Зейделя. Вектор правой части задается как $b = Ax$, где $x_i = N$.

N = 21

m = 18

$\beta = 900$

$$a_{i,j} = \frac{cos(\, i + j)}{90} + 90 \cdot e^{-(i-j)^2}$$

```
def seidel(A, b, eps):
    n = A.shape[0]
    #найдём матрицу B, B1, B2
    B = np.matrix(np.array([[-A[i, j] / A[i, i] if i != j else 0 for j in range(n)] for i in range(n)]))
    B1 = np.matrix(np.array([[B[i, j] if i > j else 0 for j in range(n)] for i in range(n)]))
    B2 = np.matrix(np.array([[B[i, j] if i < j else 0 for j in range(n)] for i in range(n)]))
    eps1 = ((1 - linalg.norm(B, ord = np.inf)) * eps) / linalg.norm(B2, ord = np.inf)
    if linalg.norm(B1, ord = np.inf) + linalg.norm(B2, ord = np.inf) < 1:
        x_ = np.array([0] * n)
        x = np.array([0] * n)
        it = 1
        while True:
            for i in range(n):
                x[i] = (b[i] / A[i, i]) + sum(B1[i, j] * x[j] for j in range(i)) + sum(B2[i, j] * x_[j] for j in range(i, n))
            if linalg.norm(x - x_, ord = 2) < eps1:
                break
            else:
                x_ = x
                it += 1
        print("Mtx B = \n", B)
        print("eps1 = ", eps1)
        print("Norm Mtx B = \n", linalg.norm(B, ord = np.inf))
        return x, it
```

```
A = np.array([
             [np.math.cos(i + j)/(90) + 90 * np.math.exp(-(i - j)**2) for j in range(10)]
             for i in range(10)]
             )
x_def = np.array([21] * 10)
b = A @ x_def
print("Mtx A = \n", A)
x, it = seidel(A, b, 10 ** (-10))
print("x = ",x_def)
print("x* = ", x)
print("iterations number = ", it)
```

```
Mtx A =
 [[ 9.00111111e+01  3.31151531e+01  1.64378365e+00  1.06965739e-04
  -7.25257873e-03  3.15180331e-03  1.06685587e-02  8.37669171e-03
  -1.61666704e-03 -1.01236696e-02]
 [ 3.31151531e+01  8.99953761e+01  3.30981498e+01  1.64114479e+00
   1.42586844e-02  1.06786869e-02  8.37669296e-03 -1.61666704e-03
  -1.01236696e-02 -9.32301699e-03]
 [ 1.64378365e+00  3.30981498e+01  8.99927373e+01  3.31123015e+01
   1.65907606e+00  1.94835741e-02 -1.60653888e-03 -1.01236683e-02
  -9.32301699e-03  4.91744221e-05]
 [ 1.06965739e-04  1.64114479e+00  3.31123015e+01  9.00106686e+01
   3.31175264e+01  1.64679083e+00  9.83212791e-04 -9.31288882e-03
   4.91756720e-05  9.37615510e-03]
 [-7.25257873e-03  1.42586844e-02  1.65907606e+00  3.31175264e+01
   8.99983833e+01  3.30990260e+01  1.63908448e+00  1.11560568e-02
   9.38628326e-03  1.00827433e-02]
 [ 3.15180331e-03  1.06786869e-02  1.94835741e-02  1.64679083e+00
   3.30990260e+01  8.99906770e+01  3.31091989e+01  1.65778366e+00
   2.11896244e-02  1.52943059e-03]
 [ 1.06685587e-02  8.37669296e-03 -1.60653888e-03  9.83212791e-04
   1.63908448e+00  3.31091989e+01  9.00093762e+01  3.31192324e+01
   1.64992680e+00  2.66590556e-03]
 [ 8.37669171e-03 -1.61666704e-03 -1.01236683e-02 -9.31288882e-03
   1.11560568e-02  1.65778366e+00  3.31192324e+01  9.00015193e+01
   3.31007087e+01  1.63776684e+00]
 [-1.61666704e-03 -1.01236696e-02 -9.32301699e-03  4.91756720e-05
   9.38628326e-03  2.11896244e-02  1.64992680e+00  3.31007087e+01
   8.99893593e+01  3.31060923e+01]
 [-1.01236696e-02 -9.32301699e-03  4.91744221e-05  9.37615510e-03
   1.00827433e-02  1.52943059e-03  2.66590556e-03  1.63776684e+00
   3.31060923e+01  9.00073369e+01]]
```

```
Mtx B =
 [[ 0.00000000e+00 -3.67900725e-01 -1.82620082e-02 -1.18836150e-06
   8.05742607e-05 -3.50157139e-05 -1.18524909e-04 -9.30628631e-05
   1.79607498e-05  1.12471332e-04]
 [-3.67965050e-01  0.00000000e+00 -3.67776115e-01 -1.82358790e-02
  -1.58437967e-04 -1.18658173e-04 -9.30791483e-05  1.79638901e-05
   1.12490997e-04  1.03594400e-04]
 [-1.82657367e-02 -3.67786899e-01  0.00000000e+00 -3.67944153e-01
  -1.84356661e-02 -2.16501627e-04  1.78518726e-05  1.12494282e-04
   1.03597438e-04 -5.46426563e-07]
 [-1.18836734e-06 -1.82327808e-02 -3.67870854e-01  0.00000000e+00
  -3.67928901e-01 -1.82955072e-02 -1.09232917e-05  1.03464278e-04
  -5.46331594e-07 -1.04167153e-04]
 [ 8.05856557e-05 -1.58432673e-04 -1.84345096e-02 -3.67979126e-01
   0.00000000e+00 -3.67773562e-01 -1.82123770e-02 -1.23958413e-04
  -1.04293910e-04 -1.12032493e-04]
 [-3.50236649e-05 -1.18664369e-04 -2.16506584e-04 -1.82995716e-02
  -3.67805057e-01  0.00000000e+00 -3.67918100e-01 -1.84217267e-02
  -2.35464662e-04 -1.69954338e-05]
 [-1.18527193e-04 -9.30646708e-05  1.78485725e-05 -1.09234486e-05
  -1.82101527e-02 -3.67841666e-01  0.00000000e+00 -3.67953138e-01
  -1.83306104e-02 -2.96180873e-05]
 [-9.30727812e-05  1.79626639e-05  1.12483305e-04  1.03474796e-04
  -1.23954094e-04 -1.84195074e-02 -3.67985260e-01  0.00000000e+00
  -3.67779444e-01 -1.81971021e-02]
 [ 1.79650911e-05  1.12498518e-04  1.03601326e-04 -5.46460963e-07
  -1.04304368e-04 -2.35468110e-04 -1.83346877e-02 -3.67829141e-01
   0.00000000e+00 -3.67888966e-01]
 [ 1.12476048e-04  1.03580634e-04 -5.46337930e-07 -1.04171009e-04
  -1.12021349e-04 -1.69922880e-05 -2.96187583e-05 -1.81959260e-02
  -3.67815486e-01  0.00000000e+00]]

eps1 =  5.86646369122494e-11
Norm Mtx B =
 0.7730671093422276
x =  [21 21 21 21 21 21 21 21 21 21]
x* =  [19 19 19 19 19 19 19 19 22 20]
iterations number =  2
```

За минимальное количество итераций получен практически правильный ответ.

Критерий окончания $\left\|x^{(n)} - x^{(n-1)}\right\| \le \frac{1-\|B\|}{\|B_2\|}\varepsilon$.