Решение систем линейных алгебраических уравнений методом LR-разложения

Описание метода:

В данной задаче имеется матричное уравнение вида *AX = F*. Буду решать его с помощью LR-разложения.

Для этого исходную матрицу A нужно привести к виду *A = LR*, где L – нижняя треугольная матрица, R – верхняя треугольная матрица.

Это можно сделать с помощью следующих формул:

$$\mathbf{L}_{*1} = \mathbf{A}_{*1},\ \mathbf{R}_{1*} = l_{11}^{-1}\mathbf{A}_{1*},$$

$$\mathbf{L}_{*k} = \mathbf{A}_{*k} - \sum_{i=1}^{k-1}\mathbf{L}_{*i} r_{ik},\ \mathbf{R}_{k*} = l_{kk}^{-1}(\mathbf{A}_{k*} - \sum_{i=1}^{k-1} l_{ki} R_{i*}),\ i = k+1,\ldots,n$$

После этого хода решается следующая задача:

$$\mathbf{A} = \mathbf{LR} \ \Rightarrow\ (\mathbf{LR})\mathbf{x} = \mathbf{f} \ \Rightarrow \left[\begin{array}{l}\mathbf{Lg} = \mathbf{f}\\ \mathbf{Rx} = \mathbf{g}\end{array}\right.$$

Нахождения g – не что иное как прямая подстановка и формулы для нее соответственно:

$$\mathbf{G}_{1*} = l_{11}^{-1}\mathbf{F}_{1*},$$

$$\mathbf{G}_{k*} = l_{kk}^{-1}(f_k - \sum_{i=1}^{k-1} l_{ki}\mathbf{G}_{i*}),\ k = 2,\ldots,n$$

Далее *X* находится обратной подстановкой:

$$\mathbf{X}_{n*} = \mathbf{G}_{n*},\ \mathbf{X}_{k*} = \mathbf{G}_{k*} - \sum_{j=k+1}^{n} r_{kj}\mathbf{X}_{j*},\ k = n-1,\ldots,1$$

Однако в данной задаче могут возникать случаи деления на ноль, так как пока что ничего не сказано про выбор текущего элемента. В связи с этим будем производить полный выбор ведущего элемента, чтобы избегать таких случаев и улучшить точность решения.

Формулы:

$$\left|a_{pq}^{(k-1)}\right| = \max_{i,j\geq k}\left|a_{ij}^{(k-1)}\right| \ \Rightarrow \text{ перестановка } \begin{cases}\text{строк} & p \Leftrightarrow k\\ \text{столбцов} & q \Leftrightarrow k\end{cases}$$

Программная реализация:

```
import numpy as np


def swap(a: np.array, i, j, axis=0):
    if axis == 0:
        for k in range(a.shape[1]):
            temp = a[i, k]
            a[i, k] = a[j, k]
            a[j, k] = temp
    if axis == 1:
        for k in range(a.shape[0]):
            temp = a[k, i]
            a[k, i] = a[k, j]
            a[k, j] = temp


def get_q(a: np.array, f: np.array):
    n = a.shape[0]
    q = [0 for i in range(n)]
    for k in range(n):
        pi, qi = k, k
        for i in range(k, n):
            if abs(a[i, k]) > abs(a[pi, k]):
                pi = i
        q[k] = int(qi)
        swap(a, k, pi, 0)
        swap(f, k, pi, 0)
    return q
```

```python
def get_lu(a: np.array):
    n = a.shape[0]
    l = np.zeros([n, n])
    u = np.zeros([n, n])
    for i in range(n):
        l[i, i] = 1

    for i in range(n):
        for j in range(n):
            s = 0
            if i <= j:
                for k in range(i):
                    s += l[i, k] * u[k, j]
                u[i, j] = (a[i, j] - s)
            else:
                for k in range(j):
                    s += l[i, k] * u[k, j]
                l[i, j] = (a[i, j] - s) / u[j, j]
    return l, u


def direct_sub(l: np.array, f: np.array):
    n = f.shape[0]
    g = np.zeros(n)
    for k in range(n):
        s = 0
        for i in range(k):
            s += l[k, i] * g[i]
        g[k] = (f[k] - s) / l[k, k]
    return g
```

```python
def back_sub(u: np.array, g: np.array):
    n = g.shape[0]
    x = np.zeros(n)
    for k in range(n - 1, -1, -1):
        s = 0
        for j in range(k + 1, n):
            s += u[k, j] * x[j]
        x[k] = (g[k] - s) / u[k, k]
    return x


def lu_solve(a: np.array, f: np.array):
    p = get_q(a, f)
    n, m = f.shape[0], f.shape[1]
    l, u = get_lu(a)
    res = np.zeros([n, m])
    for i in range(m):
        cur = np.zeros(n)
        for j in range(n):
            cur[j] = f[j, i]
        y = direct_sub(l, cur)
        x = back_sub(u, y)
        for j in range(n):
            res[j, i] = x[j]
    res = np.array([res[p[i]] for i in range(n)])
    return res
```

Проверка решения

Сначала буду брать матрицу А и матрицу Х, как матрицы, составленные из случайных чисел

Для этих случайных данных выведу матрицу L и R:

```
[[ 1.          0.          0.          0.          0.        ]
 [-0.71220734  1.          0.          0.          0.        ]
 [ 0.61932906 -0.87141513  1.          0.          0.        ]
 [-0.88060321  0.5972825  -0.43009988  1.          0.        ]
 [-0.86455543  0.03688508  0.79043257  1.17422814  1.        ]]
[[-11.8285916    4.30832548   4.37524504   6.51006423  -5.65680911]
 [  0.          15.05584796   7.87305403  -3.31091683  -9.8779017 ]
 [  0.           0.          12.44564157   2.8859865  -13.38604416]
 [  0.           0.           0.          -0.83557094  -6.28031223]
 [  0.           0.           0.           0.           7.91594772]]
```

Как видно, все работает

Попробую для матрицы с нулевой диагональю, где могули бы возникнуть трудности без полного выбора ведущего элемента

```
a = np.random.uniform(-12, 12, (5, 5))
f = np.random.uniform(-12, 12, (5, 4))
for i in range(a.shape[0]):
    a[i, i] = 0
x = lu_solve(a, f)
print(np.linalg.norm(a.dot(x) - f))
```

Результат:

```
5.651645894066188 5e-14

Process finished with exit code 0
```

Все работает

Теперь выведу норму матрицы невязки для 10 случайных наборов данных

```
for _ in range(10):
    a = np.random.uniform(-12, 12, (5, 5))
    f = np.random.uniform(-12, 12, (5, 4))
    x = lu_solve(a, f)
    print(np.linalg.norm(a.dot(x) - f))
```

Результат:

```
1.1305210860370893e-14
3.775105212478043e-14
5.9993148609410945e-15
7.52107283203973e-15
1.354472090042691e-14
8.965764119249463e-14
8.782918492649856e-15
2.2609434689158658e-14
9.256890181353153e-15
4.35435358193423e-15
```

Как видно, здесь погрешность близка к машинной погрешности, в связи с чем, можно сказать, что реализованный алгоритм работает правильно и выдает хороший результат.