实验六 线性代数方程组的直接解法

一、实验目的

1. 掌握高斯消去法的基本思路和迭代步骤；
2. 了解高斯消去法可能遇到的困难。

二、实验过程和结果

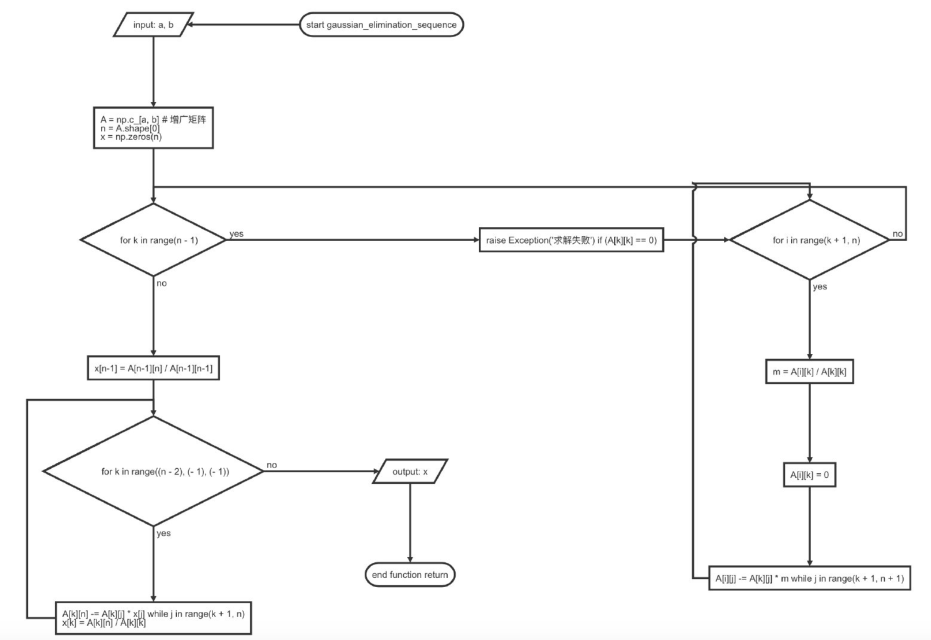
1. 顺序高斯消元法

对于方程组 $Ax = b$，高斯消元法将通过初等行变换将增广矩阵 $B = [A \vdots b]$ 中的 A 变为上三角矩阵（消元过程），然后解这个三角方程组（回代过程）。

编程实现[1]：

```python
def gaussian_elimination_sequence(a, b):
    """用「顺序高斯消去法」解线性方程 `ax = b`。

    Args:
        a : np_array_like 系数矩阵 (nxn)
        b : np_array_like 右端常数 (n)

    Returns:
        x : np.array `ax=b` 的解 (n)

    Raises:
        Exception("求解失败") if a[k][k] == 0
    """
    ...
```

具体程序流程图：



调用实例[2]：

```python
q = np.array([[0.101, 2.304, 3.555, 1.183],
              [-1.347, 3.712, 4.623, 2.137],
              [-2.835, 1.072, 5.643, 3.035]])
a, b = q[:, :-1],  q[:, -1]
gaussian_elimination_sequence(a, b)
```

```
array([-0.39823377,  0.01379507,  0.33514424])
```

```python
# 用 numpy 计算检查
np.linalg.solve(a, b)
```

```
array([-0.39823377,  0.01379507,  0.33514424])
```

2. 列主元高斯消元法

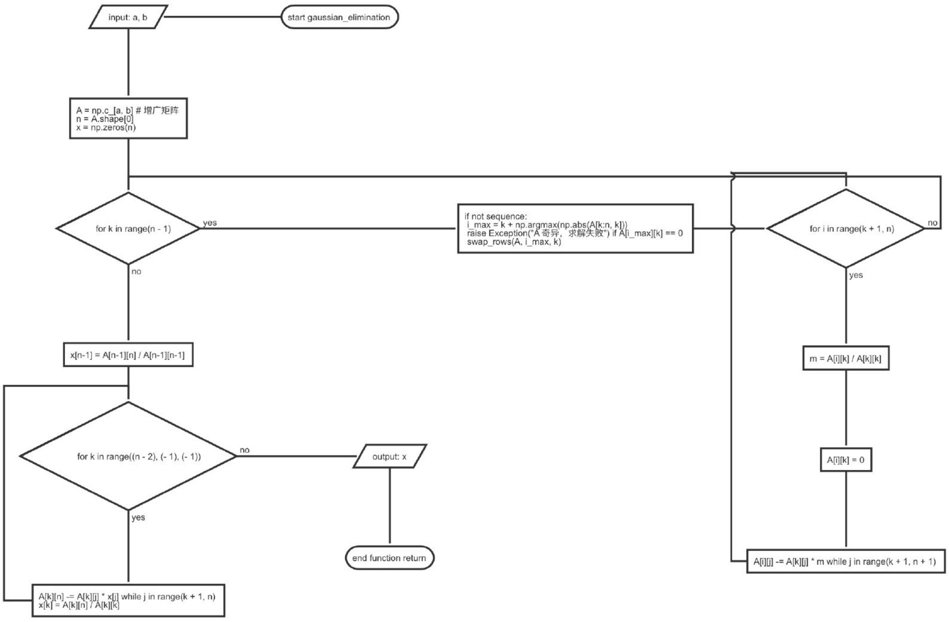
列主元高斯消元法的实现和顺序高斯消元法很类似，只是在消元的过程中加入了列选主元、行交换的过程：

```python
...
for k in range(0, n-1):
    i_max = k + np.argmax(np.abs(A[k:n, k]))  # 选主元
    A[[i_max, k]] = A[[k, i_max]]  # 行交换

    for i in range(k+1, n):  # 消元
        ...
...
```

代码实现[1]：

```python
def gaussian_elimination(a, b, sequence=False):
    """用「高斯消去法」解线性方程 `ax = b`。

    本函数可以通过「列主元高斯消元法」或「顺序高斯消元法」计算，
    通过参数 sequence 控制，默认 sequence=False 使用「列主元高斯消元法」。

    Args:
        a : np_array_like 系数矩阵 (nxn)
        b : np_array_like 右端常数 (n)

    Returns:
        x : np.array `ax=b` 的解 (n)

    Raises:
        Exception("求解失败") if a[k][k] == 0
    """
    ...
```

完整的程序流程图：



调用实例:

```
q = np.array([[0.101, 2.304, 3.555, 1.183],
              [-1.347, 3.712, 4.623, 2.137],
              [-2.835, 1.072, 5.643, 3.035]])
a, b = q[:, :-1],  q[:, -1]
print('列主元高斯消元:\t', gaussian_elimination(a, b))
print('顺序高斯消元:\t', gaussian_elimination(a, b, sequence=True))
```

```
列主元高斯消元:     [-0.39823377  0.01379507  0.33514424]
顺序高斯消元:      [-0.39823377  0.01379507  0.33514424]
```

3. LU 分解

LU 分解将系数矩阵 A 分解为一个下三角矩阵 L 和一个上三角矩阵 U 的乘积：$A = LU$。然后原方程组 $Ax = b$ 就可以通过以下两步解出：

- 解三角方程 $Ly = b$
- 解三角方程 $Ux = y$，求得的 x 即原方程组的解。

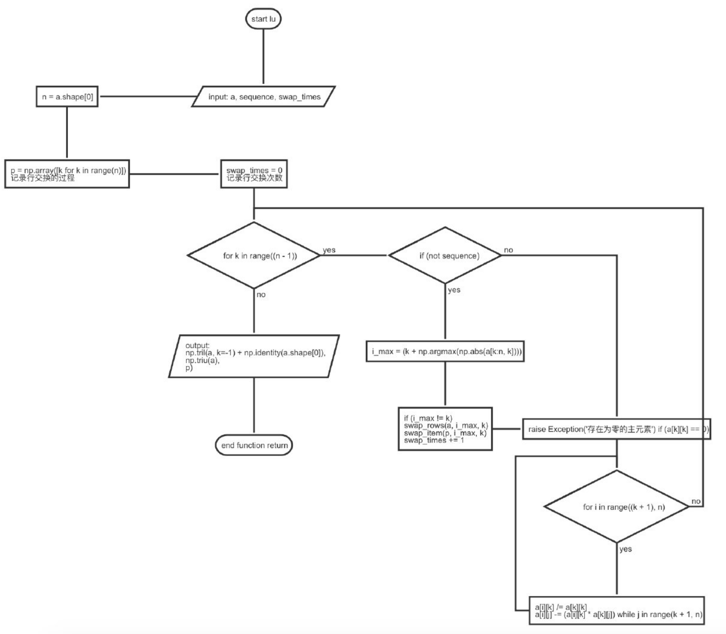
解三角方程会很容易，而且使用 LU 分解可以在 A 不变，b 取不同值时快速给出解，避免大量重复运算。

使用列主元高斯消元时，得到的 LU 分解中还有一个记录行交换的交换矩阵 P：$PA = LU$，在回代过程时要置 $b = Pb$。

编程实现[3]，由于顺序高斯消去法和列主元的高斯消去法的 LU 分解在实现上区别不大，只是列主元需要加入列选主元和行交换的过程，所以完全可以用一个函数实现这两种方法，通过一个参数控制具体方法：

```python
def lu(a, sequence=False, swap_times: list = {}):
    """LU 分解

    Args:
        a: np_array_like 系数矩阵 (nxn)
        sequence: bool, True 则使用顺序高斯消去法, False 为列主元的高斯消去法
            default: sequence=False
        swap_times: 这是一个**输出**用的变量, 只有传入 dict 变量时才有效。
            若使用「列主元高斯消元法」 (sequence=False)
            则, 置 swap_times['swap_times'] = 行交换次数。
            这个值正常的输出中不需要, 但在一些问题, 比如,
            利用 LU 分解求行列式时, 得到 swap_times 会很有帮助。

    Returns:
        (l, u, p): result

        l: np.array, Lower triangle result (nxn)
        u: np.array, Upper triangle result (nxn)
        p: np.array, Permutation: 交换后的行顺序 (n)
            p = None if sequence=True

    Raises:
        Exception: 存在为零的主元素
    """
    ...
```

程序流程图:

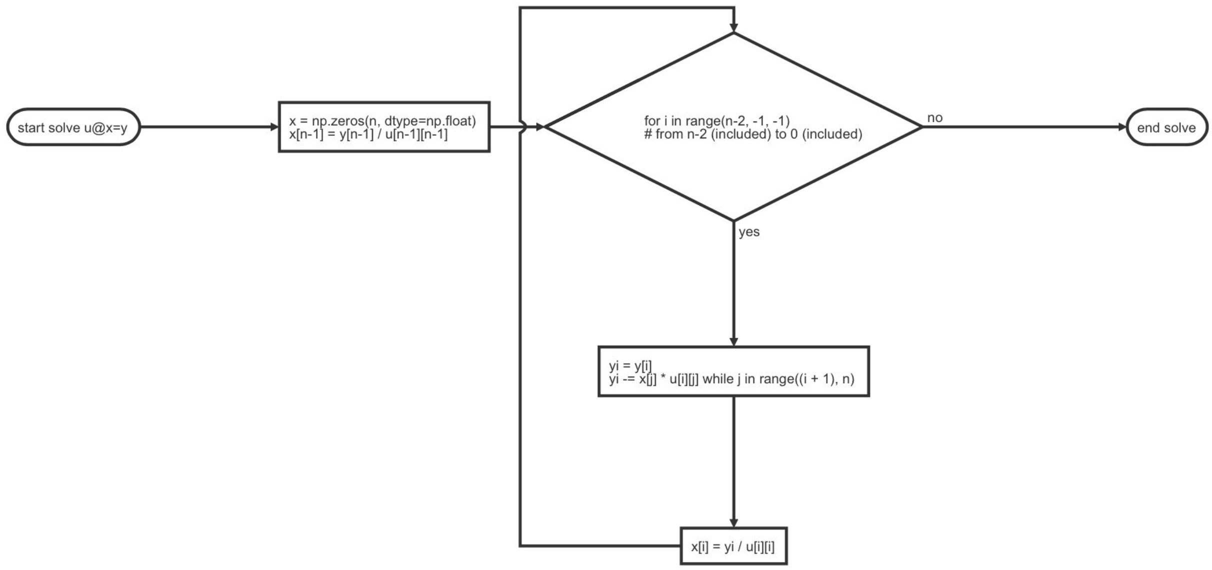


利用 LU 分解的结果解原方程组的程序实现很简单，就是解两个三角方程，编程实现:

```python
def solve_lu(b, l, u, p=None):
    """用 lu(a) 得到的 `pa=lu` 分解的结果求解原方程组 `ax=b` 的解 x。
    若 p 不为 None 则使用「列主元高斯消元」, p 为 None表示使用「顺序高斯消元」。

    Args:
        b: np_array_like, 原方程组的右端常数 (n)
        l: np_array_like, Lower triangle of lu_seq(a)
        u: np_array_like, Upper triangle of lu_seq(a)
        p: np_array_like, LU分解中交换后的行顺序
            default p=None: 未做行交换, 即使用顺序高斯消去法

        使用列主元高斯消元法时, l, u, p 使用 lu(a) 得到的结果即可:
            solve_lu(b, *lu(a))
        或者使用顺序高斯消元:
            solve_lu(b, *lu(a, sequence=True))  # p=None

```

```
    Returns:
        x : np.array `ax=b` 的解 (n)
    """
    b = p @ b if p != None
    y = solve(l, b)
    x = solve(u, y)

    return x
```

这里的 solve 是解方程的具体过程，以后一个为例：



调用实例：

```
q = np.array([[0.101, 2.304, 3.555, 1.183],
              [-1.347, 3.712, 4.623, 2.137],
              [-2.835, 1.072, 5.643, 3.035]])
a, b = q[:, :-1],  q[:, -1]

# 列主元高斯消元
x = solve_lu(b, *lu(a)); print(x)
# 顺序高斯消元
x = solve_lu(b, *lu(a, sequence=True)); print(x)
```

```
[-0.39823377  0.01379507  0.33514424]
[-0.39823377  0.01379507  0.33514424]
```

三、思考题分析解答

1. 解方程：$A = \begin{bmatrix} 0.3\times10^{-15} & 59.14 & 3 & 1 \\ 5.29 & -6.13 & -1 & 2 \\ 11.2 & 9 & 5 & 2 \\ 1 & 2 & 1 & 1 \end{bmatrix}$，$b = [59.17, 46.78, 1, 2]$。

解[4]：

```python
A = [[0.3e-15, 59.14, 3,1],
     [5.29, -6.13, -1, 2],
     [11.2, 9, 5, 2],
     [1, 2, 1, 1]]
b = [59.17, 46.78, 1, 2]
```

参考的正确解:

```python
np.linalg.solve(A, b)
```

```
array([  3.84604049,   1.60956057, -15.47712511,  10.41196349])
```

顺序高斯消去:

```python
gaussian_elimination(A, b, sequence=True)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:36: RuntimeWarning: inva
lid value encountered in double_scalars
array([nan, nan, nan, nan])
```

列主元高斯消元:

```python
gaussian_elimination(A, b, sequence=False)
```

```
array([  3.84604049,   1.60956057, -15.47712511,  10.41196349])
```

这里由于$A_{0,0}$ 的值非常小，顺序法使用这个值去消元造成了精度丢失，并且超出了 double 类型能保证的精度范围，造成了结果全为 `nan`。用列主元法就避免了这种情况，正确完成了求解：

使用顺序的 LU 分解同样无法完成求解：

```python
l, u, p = lu(A, sequence=True); print(f'l={l}\nu={u}\np={p}')
x = solve_lu(b, l, u, p); print(f'x={x}')
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-29-a2c3103a4e3e> in <module>
----> 1 l, u, p = lu(A, sequence=True); print(f'l={l}\nu={u}\np={p}')
      2 x = solve_lu(b, l, u, p); print(f'x={x}')

<ipython-input-3-d83f8c6bb23c> in lu(a, sequence, swap_times)
     53
     54             if a[k][k] == 0:
---> 55                 raise Exception("存在为零的主元素")
     56
     57             for i in range(k+1, n):

Exception: 存在为零的主元素
```

列主元的 LU 分解可以正确求解：

```
l, u, p = lu(A); print(f'l={l}\nu={u}\np={p}')
x = solve_lu(b, l, u, p); print(f'x={x}')
```

```
l=[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 2.67857143e-17  1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 4.72321429e-01 -1.75530823e-01  1.00000000e+00  0.00000000e+00]
 [ 8.92857143e-02  2.02304459e-02 -1.73854511e-01  1.00000000e+00]]
u=[[11.2         9.          5.          2.        ]
 [ 0.         59.14        3.          1.        ]
 [ 0.          0.         -2.83501467  1.23088797]
 [ 0.          0.          0.          1.01519355]]
p=[2 0 1 3]
x=[  3.84604049   1.60956057 -15.47712511  10.41196349]
```

2. 计算方阵的行列式

利用 LU 分解求解线性方程组:

$$\det A = \det(LU) = (\det L)(\det U) \tag{1}$$

L 和 U 都是三角矩阵，其行列式的值等于对角线元素乘积。所以这里 $\det L = 1$ ，$\det A = \prod diag(U)$。

编程实现:

```
def det(a):
    swap_times = {}
    l, u, p = lu(a, sequence=False, swap_times=swap_times)
    sign = -1 if swap_times['swap_times'] % 2 == 1 else 1
    return np.prod(np.diag(u)) * sign
```

由于顺序高斯消元的一些缺陷，这里使用了列主元的高斯消元，所以在计算时有行交换的情况，每交换一次行列式的值就要乘上 -1。

3. 计算矩阵的逆

利用LU分解，也可以完成求逆。记 A 的逆矩阵为 X，I 为单位矩阵。有:

$$AX = I \tag{2}$$

按列分块 $X = (x_0, \ldots, x_n)$，$I = (e_0, \ldots, e_n)$，则:

$$A(x_0, \ldots, x_n) = (e_0, \ldots, e_n) \tag{3}$$

对 A 做 LU 分解，逐次取 $b = e_i \quad (i = 0, \ldots, n)$，解 $Ax_i = b$，即可求得逆 X。

编程实现:

```
def inv(a):
    l, u, p = lu(a)

    X = np.zeros_like(a, dtype=np.float)
    B = np.identity(np.shape(a)[0])

    for i, b in enumerate(B.T):  # iter on cols
        x = solve_lu(b, l, u, p)
        X[:, i] = x

    return X
```

四、重点难点分析

重点：

1. 掌握高斯消去法的基本思路和迭代步骤
2. 了解高斯消去法可能遇到的困难

难点：

1. 利用 LU 分解做矩阵求逆的方法的推导，使用矩阵分块的思想。
2. 本实验中编程实现的所有算法在速度上均表现不佳。在一次实际的比较中[5]，对一个 150×150 的矩阵求逆，NumPy 中的实现只需 0.005 秒，而我实现的代码需要 3.866 秒。这里的实现在解方程时逐个元素去运算，并没有利用上现代计算机的并发运算能力以及 SSE 等可在此处利用的底层优化方法。但并发运算会大幅增加这里编程的难度，不考虑实现。而利用 SSE 指令运算的实现也十分艰难，所以这里的效率问题就暂时无法解决了。

---

1. 顺序高斯消元法、列主元高斯消元法实现源码: https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex6/src/gaussian_elimination.py ↩ ↩
2. 程序调用实例源码：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex6/src/ex6.ipynb ↩
3. LU分解的实现源码：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex6/src/lu.py ↩
4. 解题的具体过程见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex6/src/ex6.ipynb ↩
5. 算法速度的对比见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex6/src/ex6.ipynb ↩