实验七 线性方程组的迭代解法

一、实验目的

1. 掌握解线性方程组的雅可比迭代和高斯-塞德尔迭代算法；
2. 初步掌握解线性方程组的迭代算法的设计方法。

二、实验过程和结果

有唯一解的非奇异线性方程组 $Ax = b$，可以等价为不动点方程 $x = Bx + f$，由此建立迭代公式：

$$x^{(k+1)} = Bx^{(k)} + f \qquad k = 0, 1, 2, \dots \tag{1}$$

通过特定方式构建 B 和 f，给定初始向量 $x^{(0)}$ 即可迭代解出原方程的数值近似解。

1. 雅可比迭代法

把系数矩阵 A 分裂成 $A = D - L - U$，其中 $D = diag(a_{11}, \dots, a_{nn})$，$-L$ 和 $-U$ 分别是 A 的下三角和上三角部分。则 Jacobi 迭代法就是取 $B = D^{-1}(L + U), \quad f = D^{-1}b$:

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b, \qquad k = 0, 1, 2, \dots \tag{2}$$

编程实现（这里只保留函数声明头和简化的文档注释，具体实现源码见https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/jacobi_iter.py）：

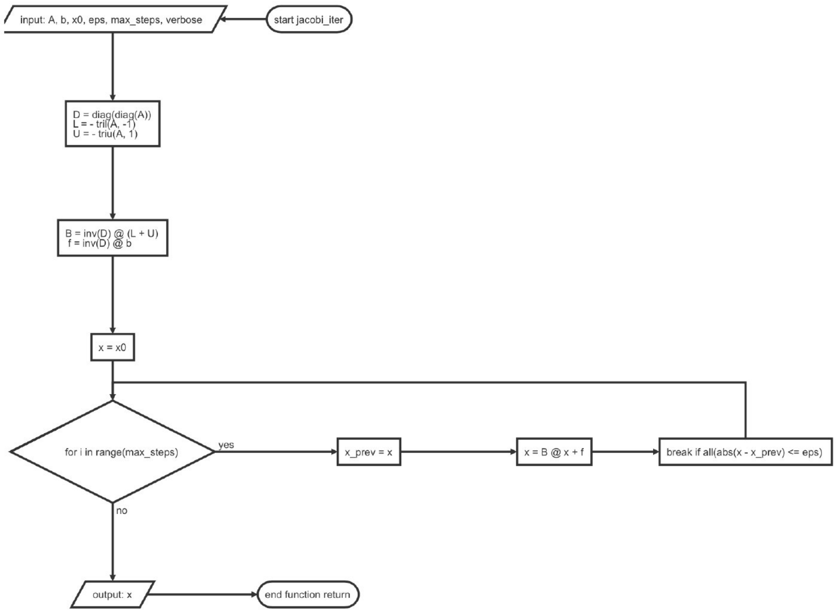
```python
def jacobi_iter(A, b, x0=None, eps=1e-5, max_steps=5000, verbose=False):
    """雅可比（Jacobi）迭代法求解线性方程组：A @ x = b

    Args:
        A:  np_array_like，系数矩阵
        b:  np_array_like，右端常数
        x0: np_array_like，迭代初值 default x0=None: use a random array.
        eps: float，精度要求
        max_steps: int，最大迭代次数
        verbose: bool，如果计算成功，打印出结果及迭代次数

    Returns:
        x: 方程组的解

    Raises:
        ValueError: 参数 A, b 和 x0 存在形状不匹配
        Expection:  达到最大迭代次数，仍不满足精度
    """
    ...
```

具体的程序流程图：



调用测试：

```
jacobi_iter([[9,-1,-1], [-1,10,-1],[-1,-1,15]], [7,8,13], verbose=True)
```

```
jacobi_iter get result x = [0.99999906 0.99999911 0.99999933] after 7 iterations.
```

2. 高斯-塞德尔迭代法

与 Jacobi 迭代类似，做 $A = D - L - U$ 的分解之后，Gauss Seidel 迭代表示为：

$$x^{(k+1)} = (D-L)^{-1}Ux^{(k)} + (D-L)^{-1}b, \qquad k = 0, 1, 2, \ldots \tag{3}$$

也就是 (1) 式取 $B = (D-L)^{-1}U$，$f = (D-L)^{-1}b$ 的情形。

编程实现和 Jacobi 迭代类似，只需修改 B 和 f 的求解方法：

```python
def gauss_seidel_iter(A, b, x0=None, eps=1e-5, max_steps=5000, verbose=False):
    ...
    inv_DsL = np.linalg.pinv(D - L)

    B = inv_DsL @ U
    f = inv_DsL @ b
    ...
```

（完整代码实现见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/gauss_seidel_iter.py）

调用测试：

```
gauss_seidel_iter([[9,-1,-1], [-1,10,-1],[-1,-1,15]], [7,8,13], eps=1e-12, verbose=True)
```

```
gauss_seidel_iter get result x = [1. 1. 1.] after 10 iterations.
```

3. 逐次超松弛迭代法

逐次超松弛（Successive over-relaxation，SOR）迭代法引入一个松弛因子 $\omega > 0$，迭代方程：

$$x^{(k+1)} = (D-\omega L)^{-1}\left((1-\frac{1}{\omega})D+\omega U\right)x^{(k)} + \omega(D-\omega L)^{-1}b, \qquad k=0,1,2,\dots \qquad (4)$$

也就是 (1) 式取 $B=(D-\omega L)^{-1}\left((1-\frac{1}{\omega})D+\omega U\right)$，$f=\omega(D-\omega L)^{-1}b$ 的情形。

编程实现依然和 Jacobi 迭代类似，只需修改 B 和 f 的求解方法，此处不在赘述。

调用测试：

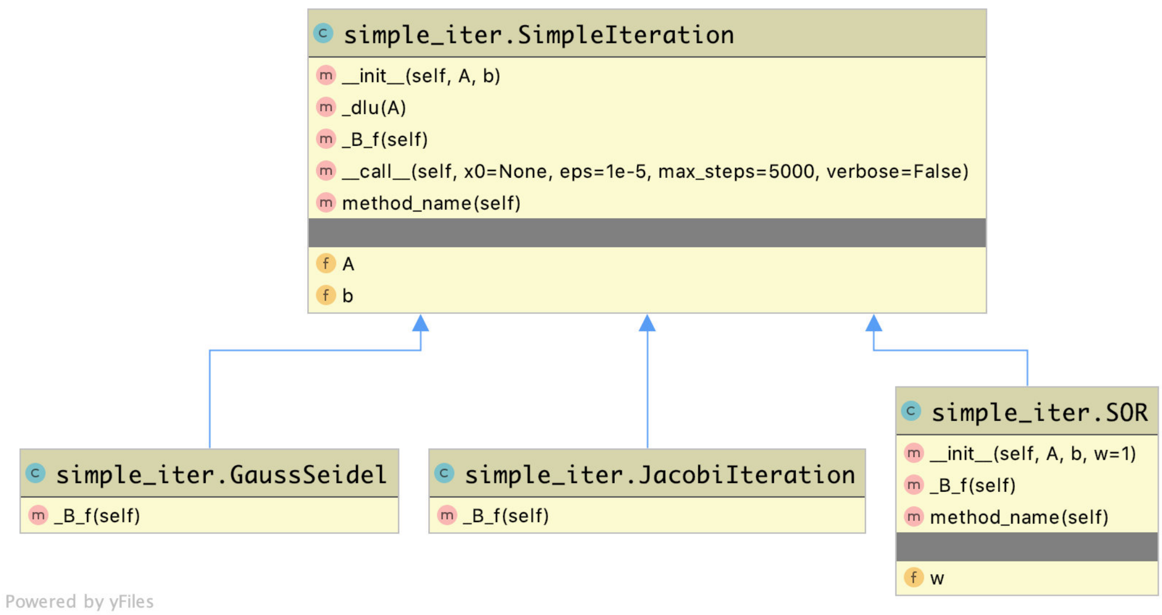
```
sor([[9,-1,-1], [-1,10,-1],[-1,-1,15]], [7,8,13], w=1.1, eps=1e-12, verbose=True)
```

```
sor (w=1.1) get result x = [1. 1. 1.] after 14 iterations.
```

（完整代码实现见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/sor.py）

4. 三个算法的封装与比较

上述的 Jacobi 迭代法、Gauss Seidel 迭代法、SOR 迭代法的代码实现相当类似，仅有 B、f 的计算方法有差异，所以完全可以用简单的面向对象方法进行封装，避免代码重复：



（具体的代码实现见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/simple_iteration.py）

对于实验内容中的问题分别使用 Jacobi、Gauss-Seidel 迭代以及取不同的松弛因子的 SOR，在相同的初值、精度要求下，得到的计算结果如下：

```
A = [[-4, 1, 1, 1],
     [1, -4, 1, 1],
     [1, 1, -4, 1],
     [1, 1, 1, -4]]
b = [1, 1, 1, 1]

x0 = [0, 0, 0, 0]

for method in [JacobiIteration(A, b), GaussSeidel(A, b), SOR(A, b, 0.9), SOR(A, b, 1), SOR(A, b, 1.1)]:
    method(x0=x0, eps=1e-9, verbose=True)
```

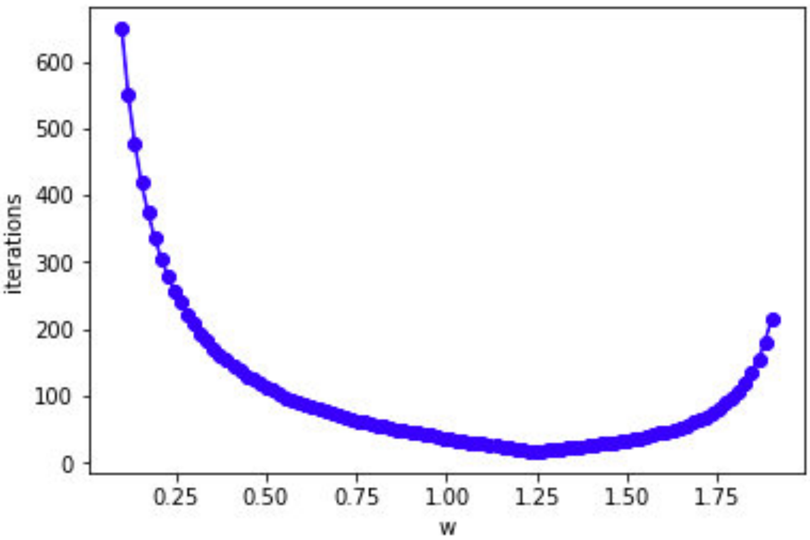
```
JacobiIteration get result x = [-1. -1. -1. -1.] after 68 iterations.
GaussSeidel get result x = [-1. -1. -1. -1.] after 36 iterations.
SOR (w=0.9) get result x = [-1. -1. -1. -1.] after 46 iterations.
SOR (w=1) get result x = [-1. -1. -1. -1.] after 36 iterations.
SOR (w=1.1) get result x = [-1. -1. -1. -1.] after 28 iterations.
```

可以看到，Gauss-Seidel 在收敛速度上优于 Jacobi 迭代。

取不同松弛因子会对 SOR 的收敛速度有影响，可以进一步实验来寻找一个较优的松弛因子：

```
for w in np.linspace(0.1, 1.9, 100):
    rs = SOR(A, b, w)(x0=[0, 0, 0, 0], eps=1e-9, verbose=True)
```

通过一系列计算，得到不同松弛因子取值与迭代次数的关系，如下图所示：



容易确定其中最优的迭代次数为 `17` 次，对应的松弛因子为 `1.2272727272727273` 或 `1.2454545454545456`，最优的松弛因子应该在这两个值之间，容易利用二分法进一步逼近最优松弛因子。这里我认为进一步试算意义不大了，故不在继续。

（具体求解过程源码见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/ex7.ipynb）

三、思考题分析解答

（1）

试用前面程序计算如下方程组

$$\begin{cases} x_1 - 3x_2 - 6x_3 = 1 \\ 2x_1 + 8x_2 - 3x_3 = 21 \\ 5x_1 + 2x_2 + x_3 = 8 \end{cases}$$

写出你遇到的困难和解决办法。

解：

首先把问题录入计算机，并使用已有工具求出一个参考的解：

```
A = [[1, -3, -6],
     [2, 8, -3],
     [5, 2, 1]]
b = [1, 21, 8]
np.linalg.solve(A, b)
```

```
array([ 1.,  2., -1.])
```

直接利用 Gauss-Seidel 求解，会遇到无法收敛的问题：

```
gauss_seidel_iter(A, b, eps=1e-8, verbose=True)
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:48: RuntimeWarning: overflow encountered in matmul
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:48: RuntimeWarning: invalid value encountered in matmul
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:50: RuntimeWarning: invalid value encountered in less_equal
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-53-1c82c63562b5> in <module>
----> 1 gauss_seidel_iter(A, b, eps=1e-8, verbose=True)

<ipython-input-45-6fde854fad32> in gauss_seidel_iter(A, b, x0, eps, max_steps, verbose)
     51                 break
     52         else:
---> 53             raise Exception(f"cannot reach eps ({eps}) after max_steps ({max_steps}). The last result: x = {x}")
     54 
     55 

Exception: cannot reach eps (1e-08) after max_steps (5000). The last result: x = [nan nan nan]
```

手动做一些行变换，使其严格对角占优，就可以解了：

```
A0 = [[5, 2, 1],
      [2, 8, -3],
      [1, -3, -6]]
b0 = [8, 21, 1]
gauss_seidel_iter(A1, b1, eps=1e-8, verbose=True)
```

```
gauss_seidel_iter get result x = [ 1.  2. -1.] after 12 iterations.
array([ 1.,  2., -1.])
```

（具体求解过程源码见：https://github.com/cdfmlr/NumericalAnalysis/blob/master/ex7/src/ex7.ipynb）

（2）

对于线性方程组 $Ax = b$，讨论雅可比（Jacobi）迭代法和高斯-塞德尔(Gauss-Seidel)迭代法的收敛性，分别取系数矩阵

$$A = \begin{bmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix} 和 \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{bmatrix},$$

并由此说明两种迭代法在收敛性上有没包含关系。

解：

首先把问题录入计算机（随意取 $b = [1, 1, 1]$），并求出参考的解：

```
A1 = [[1, 2, -2], [1, 1, 1], [2, 2, 1]]

A2 = [[2, -1, 1], [2, 2, 2], [-1, -1, 2]]

b = [1, 1, 1]
```

```
np.linalg.solve(A1, b)
```

```
array([-3.,  3.,  1.])
```

```
np.linalg.solve(A2, b)
```

```
array([ 0.16666667, -0.16666667,  0.5       ])
```

分别用 Jacobi 和 Gauss-Seidel 迭代计算 $A_1 x = b$， Jacobi 迭代正确的到了结果，而 Gauss-Seidel 方法没有收敛：

```
print('jacobi: ', end='')
j = jacobi_iter(A1, b, verbose=True)
print('gauss_seidel: ', end='')
g = gauss_seidel_iter(A1, b, verbose=True)
```

```
jacobi: jacobi_iter get result x = [-3.  3.  1.] after 3 iterations.
gauss_seidel:
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-65-a8fd089a47a0> in <module>
      2 j = jacobi_iter(A1, b, verbose=True)
      3 print('gauss_seidel: ', end='')
----> 4 g = gauss_seidel_iter(A1, b, verbose=True)

<ipython-input-45-6fde854fad32> in gauss_seidel_iter(A, b, x0, eps, max_steps, verbose)
     51                 break
     52         else:
---> 53             raise Exception(f"cannot reach eps ({eps}) after max_steps ({max_steps}). The last result: x = {x}")
     54 
     55 

Exception: cannot reach eps (1e-05) after max_steps (5000). The last result: x = [nan nan nan]
```

计算 $A_2 x = b$ 时则相反， Jacobi 迭代不收敛，而 Gauss-Seidel 可以计算出结果：

```
print('gauss_seidel: ', end='')
# g = gauss_seidel_iter(A2, b, verbose=True)
print('jacobi: ', end='')
j = jacobi_iter(A2, b, verbose=True)
```

```
gauss_seidel: gauss_seidel_iter get result x = [ 0.16666863 -0.16666896  0.49999983] after 19 iterations.
jacobi:
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-67-15461764c0a3> in <module>
      2 g = gauss_seidel_iter(A2, b, verbose=True)
      3 print('jacobi: ', end='')
----> 4 j = jacobi_iter(A2, b, verbose=True)

<ipython-input-4-d9fdc2d22bab> in jacobi_iter(A, b, x0, eps, max_steps, verbose)
     52             break
     53     else:
---> 54         raise Exception(f"cannot reach eps ({eps}) after max_steps ({max_steps}). The last result: x = {x}")
     55 
     56     if verbose:

Exception: cannot reach eps (1e-05) after max_steps (5000). The last result: x = [7.99615552e+241 1.08514261e+242 2.57044248e+241]
```

四、重点难点分析

1. 掌握解线性方程组的雅可比迭代和高斯-塞德尔迭代算法；
2. 初步掌握解线性方程组的迭代算法的设计方法。