中山大學
SUN YAT-SEN UNIVERSITY

本科生实验报告

实验课程 中山大学 2021 学年春季数值计算课程

实验名称 $Gauss-Seidel$ 和 SOR 迭代法的 Python 实现

专业名称 计算机科学与技术（超算）

学生姓名 黄玟瑜

学生学号 19335074

任课教师 胡建方

实验地点

实验成绩

二〇二一年四月十一日

一、实验要求

用 Matlab 或 Python 实现 Gauss-Seidel 和 SOR 迭代法；随机生成矩阵（三角占优和非三角占优），验证算法的收敛性，并研究 SOR 中的 w 对算法结果的影响。

写成实验报告，并附上代码。

二、实验内容

实验原理

$Gauss-Seidel$ 迭代法

记 $A=D+L+U$ Jacobi 迭代形式：

$$
\begin{aligned}
&x_0 = \text{初始向量}, \\
&x_{k+1} = D^{-1}(b-(L+U)x_k), k=0,1,2,\cdot\cdot\cdot.
\end{aligned}
$$

在 Jacobi 比迭代法中，并没有对新算出的分量进行充分利用，一般来说，这些新算出计算的结果要比上一步计算的结果精确。

根据这种思路建立的迭代格式，就是高斯-赛戴尔迭代法。

$$(L+U)x_{k+1} = -Ux_k + b$$

迭代形式：

$$
\begin{aligned}
&x_0 = \text{初始向量}, \\
&x_{k+1} = D^{-1}(b-Ux_k-Lx_{k+1}), k=0,1,2,\cdot\cdot\cdot.
\end{aligned}
$$

迭代公式：

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}}{a_{ii}}$$
$$(i=1,2,...,n;k=0,1,2,...,)$$

其中 $x_i^{(k+1)}$ 表示第 k 次迭代时向量 x 的第 i 个元素。

SOR 迭代法

逐渐超松弛（Successive Over-Relaxation，SOR）的方法采取 Gauss-Seidel 趋向于解的方向并试图加速收敛。设 ω 是一实数，定义新估计量 x_{k+1} 的每个分量为 ω 乘上 Gauss-Seidel 公式与 $1-\omega$ 乘上当前估计量的加权平均。数 ω 叫做松弛参数（relaxation parameter），$\omega>1$ 时被认为是超松弛的（over-relaxation）。
迭代形式：

$$
\begin{aligned}
&x_0 = \text{初始向量}, \\
&x_{k+1} = (\omega L+D)^{-1}[(1-\omega)Px_k - \omega(x_k)] + \omega(D+\omega L)^{-1}b, k=0,1,2,\cdot\cdot\cdot.
\end{aligned}
$$

迭代公式：

$$x_i^{(k+1)} = \frac{\omega b_i - \omega\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \omega\sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + (1-\omega)a_{ii}x_i^{(k)}}{a_{ii}}$$
$$(i=1,2,...,n;k=0,1,2,...,)$$

其中 $x_i^{(k+1)}$ 表示第 k 次迭代时向量 x 的第 i 个元素。

代码解释

用 Python 编写程序，完整代码如下：

```python

import math
import numpy as np
from numpy import *


def Reform(a, b): # 列主元
    A_b = np.column_stack((a, b)) # 将两个矩阵a, b按列合并
    row = A_b.shape[0] # 初始化row为矩阵[a b]的阶数
    for i in range(0, row):
        if i < row:
            maxRow = np.argmax(abs(A_b[i:, i]))     # 找到最大元素对应的行
        else:  # 最后一个元素不用找最大值
            maxRow = 0
        b1 = maxRow + i              # 只找对角线下方的主元
        temp = np.copy(A_b[b1, :]) # 下面几行是主元和对角线所在行交换
        A_b[b1, :] = A_b[i, :]
        A_b[i, :] = temp

    return A_b


def GaussS(a, b, x):
    A_b = Reform(a, b)     # 预处理, 将每一列最大值移到对角线, 这里的p最后一行是b
    row = A_b.shape[0]     # 获取行数
    a0 = A_b[:, 0:row]      # 系数矩阵
    b0 = A_b[:, row]        # b矩阵
    iterations = 0
    err = 100.        # 初始化

    print("")
    print("Gauss-Seidel method:")

    while(err > 1.e-7 and iterations < 1000): # 控制迭代次数
        i = 0
        while(i < x.size): # 控制循环次数
            if a0[i, i] == 0:
                print('a[i, i] = 0, i = ', i)
            x[i] = (b0[i] - np.dot(a0[i, :], x) + a0[i, i] * x[i]) / a0[i, i]
            i = i+1
        iterations = iterations+1
        err = Norm(a0, b0, x) # 计算ax-b二范数
# print(iterations, x)
    return x, iterations


def Relax(a, b, x, omega):
    A_b = Reform(a, b)     # 预处理, 将每一列最大值移到对角线, 这里的p最后一行是b
    row = A_b.shape[0]     # 获取行数
    a0 = A_b[:, 0:row]      # 系数矩阵
    b0 = A_b[:, row]        #
    err = 100.
```

```python
    iterations = 0

    print("")
    print("SOR method:")

    while(err > 1.e-7 and iterations < 1000):
        i = 0
        while(i < x.size):  # 控制循环次数
            if a0[i, i] == 0:
                print('a[i, i] = 0, i = ', i)
            x[i]=(1-omega)*x[i]-omega*(np.dot(a0[i,:],x)-b0[i]-a0[i,i]*x[i])/a0[i,i]
            i = i + 1
        iterations = iterations+1
        err = Norm(a0, b0, x)  # 计算ax-b二范数
# print(iterations, x)
    return x, iterations


def Norm(a, b, x):  # 计算范数
    axb = np.dot(a, x)-b
    normaxb = np.linalg.norm(axb, ord=2)  # 二范数
    return normaxb

# 主程序


n = 7  # 设置阶数为7
# a = np.array([
#  [9, 3, 4, 0, 0, 0, 0],
#  [1, 5, 1, 0, 0, 0, 0],
#  [0, 3, 8, 2, 0, 0, 0],
#  [0, 0, 1, 7, 3, 0, 0],
#  [0, 0, 0, 2, 6, 3, 0],
#  [0, 0, 0, 0, 1, 3, 1],
#  [0, 0, 0, 0, 0, 1, 4]
# ])
# print(sum(abs(a[0, :])))

a = np.random.randint(0, 10, (n, n))  # 随机生成矩阵

# 生成严格对角占优矩阵, 若不需要该条件则注释掉以下部分代码
i = 0
while(i < n):  # 控制循环次数
    # 对每一行对角线上的元素, 使它大于所在行其他元素的绝对值之和
    while(abs(a[i, i]) < sum(abs(a[i, :]))-abs(a[i, i])):
        a[i, i] = a[i, i] + 1
    i = i + 1

answer = np.random.randint(-10, 10, (n, 1)) # 随机生成问题的正确解, 以利于后面作比较
b = np.dot(a, answer)

x = np.zeros(n, dtype=float)
print('A=')
print(a)
print('Answer=')
print(answer)
```

```
print('b=')
print(b)

x1, iterations = GaussS(np.copy(a), np.copy(b), np.copy(x))
print('GS method iterations = ', iterations)
print('x = ')
print(x1)

omega = 1.7
x2, iterations = Relax(np.copy(a), np.copy(b), np.copy(x), omega)
print('SOR method iterations = ', iterations)
print('x = ')
print(x2)
```

函数说明如下：

- **Reform** Reform 函数将系数矩阵 A 和等式右侧矩阵 b 进行适当的行交换，使其对角占优化
- **GaussS** Gauss-Seidel 迭代法求解方程组，输入矩阵 A、等式右侧矩阵 b 和初始化为 0 的向量 x，输出待求向量 x
- **Relax** SOR 迭代法求解方程组，其中 omega 为松弛因子
- **Norm** 计算 Ax-b 的二范数，二范数指空间上两个向量矩阵的直线距离，用其衡量计算结果的误差

由 Gauss-Seidel 迭代公式：

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

$$(i = 1, 2, ..., n; k = 0, 1, 2, ...,)$$

得到 Gauss-Seidel 迭代部分代码如下：

```
    while(err > 1.e-7 and iterations < 1000):  # 控制迭代次数
        i = 0
        while(i < x.size):  # 控制循环次数
            if a0[i, i] == 0:
                print('a[i, i] = 0, i = ', i)
            x[i] = (b0[i] - np.dot(a0[i, :], x) + a0[i, i] * x[i]) / a0[i, i]
            i = i+1
        iterations = iterations+1
        err = Norm(a0, b0, x)  # 计算ax-b二范数
```

若对角线元素 a0[i, i] 为 0 则输出错误，由于每次计算 x[i] 时已将结果写回，因此 $b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}$ 可由 b0[i] - np.dot(a0[i, :], x) + a0[i, i] * x[i] 获得。

计算完成后判断误差是否小于给定值，若小于则跳出循环，否则继续迭代。

控制循环次数为 1000，在一些情况下（非严格对角占优）迭代结果不会收敛，因此需要控制迭代次数。

由 SOR 迭代公式：

$$x_i^{(k+1)} = \frac{\omega b_i - \omega \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \omega \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} + (1-\omega) a_{ii} x_i^{(k)}}{a_{ii}}$$

$$(i = 1, 2, ..., n; k = 0, 1, 2, ...,)$$

得到 SOR 迭代部分代码如下：

```
    while(err > 1.e-7 and iterations < 1000):
        i = 0
        while(i < x.size):  # 控制循环次数
            if a0[i, i] == 0:
                print('a[i, i] = 0, i = ', i)
            x[i]=(1-omega)*x[i]-omega*(np.dot(a0[i,:],x)-b0[i]-a0[i,i]*x[i])/a0[i,i]
            i = i + 1
        iterations = iterations+1
        err = Norm(a0, b0, x)  # 计算ax-b二范数
```

生成非严格对角占优矩阵，调用 numpy 库函数 random.randinit：

```
a = np.random.randint(0, 10, (n, n))
```

由于没有直接生成严格对角占优矩阵的方法，因此考虑将随机生成的非严格对角占优矩阵对角占优化，将矩阵主对角元放大：

```
a = np.random.randint(0, 10, (n, n))  # 随机生成矩阵

# 生成严格对角占优矩阵，若不需要该条件则注释掉以下部分代码
i = 0
while(i < n):  # 控制循环次数
    # 对每一行对角线上的元素，使它大于所在行其他元素的绝对值之和
    while(abs(a[i, i]) < sum(abs(a[i, :]))-abs(a[i, i])):
        a[i, i] = a[i, i] + 1
    i = i + 1
```

三、实验结果

算法收敛性

严格对角占优矩阵

随机生成严格对角占优矩阵并用 Gauss-Seidel 和 SOR 迭代法求解，结果如下：

```
A=
[[35  5  8  2  7  8  5]
 [ 9 41  9  9  0  5  9]
 [ 1  4 22  6  3  8  0]
 [ 2  0  5 28  3  9  9]
 [ 8  5  0  1 25  8  3]
 [ 7  5  5  1  1 27  8]
 [ 2  4  6  7  2  8 29]]
Answer=
[[ -1]
 [  8]
 [ -9]
 [-10]
 [ -7]
 [  6]
 [ -7]]
b=
[[-123]
 [ 115]
 [-200]
 [-357]
 [-126]
 [  77]
 [-263]]

Gauss-Seidel method:
GS method iterations =  21
x =
[ -1.   8.  -9. -10.  -7.   6.  -7.]

SOR method:
SOR method iterations =  34
x =
[ -1.   8.  -9. -10.  -7.   6.  -7.]
```

```
A=
[[26  7  6  2  1  7  3]
 [ 1 21  1  1  5  7  6]
 [ 3  8 34  5  8  1  9]
 [ 7  3  4 29  7  2  6]
 [ 1  9  4  2 20  4  0]
 [ 1  0  3  9  1 16  2]
 [ 0  0  1  5  4  8 18]]
Answer=
[[ 3]
 [-2]
 [ 8]
 [ 6]
 [ 2]
 [ 1]
 [ 6]]
b=
[[151]
 [ 28]
 [366]
 [273]
 [ 73]
 [111]
 [162]]

Gauss-Seidel method:
GS method iterations =  13
x =
[ 3. -2.  8.  6.  2.  1.  6.]

SOR method:
SOR method iterations =  22
x =
[ 3. -2.  8.  6.  2.  1.  6.]
```

```
A=
[[25  7  5  4  4  2  3]
 [ 3 29  8  4  4  9  1]
 [ 3  5 31  7  8  3  5]
 [ 0  5  4 30  5  9  7]
 [ 1  6  0  7 23  0  9]
 [ 0  3  3  1  5 12  0]
 [ 3  3  0  2  4  8 20]]
Answer=
[[ 6]
 [-1]
 [-3]
 [-3]
 [ 1]
 [-5]
 [ 4]]
b=
[[ 122]
 [ -84]
 [ -88]
 [-119]
 [  38]
 [ -70]
 [  53]]

Gauss-Seidel method:
GS method iterations =  14
x =
[ 6. -1. -3. -3.  1. -5.  4.]

SOR method:
SOR method iterations =  29
x =
[ 6. -1. -3. -3.  1. -5.  4.]
```

```
A=
[[4 6 8 9 1 9 8]
 [5 1 2 4 8 1 8]
 [6 0 1 2 0 2 1]
 [1 9 5 8 8 9 1]
 [2 9 2 5 8 2 1]
 [3 4 7 9 0 7 0]
 [1 1 8 9 5 5 3]]
Answer=
[[-3]
 [ 7]
 [-8]
 [ 0]
 [-9]
 [-5]
 [ 0]]
b=
[[ -88]
 [-101]
 [ -36]
 [ -97]
 [ -41]
 [ -72]
 [-130]]

Gauss-Seidel method:
GS method iterations =  351
x =
[-2.99999999e+00  7.00000001e+00 -7.99999997e+00 -1.16947629e-08
 -9.00000000e+00 -5.00000002e+00 -2.36441622e-09]

SOR method:
SOR method iterations =  177
x =
[-2.99999999e+00  7.00000001e+00 -7.99999995e+00 -3.16032329e-08
 -9.00000000e+00 -5.00000002e+00 -2.71133966e-09]
```

非严格对角占优矩阵

随机生成非严格对角占优矩阵并用 Gauss-Seidel 和 SOR 迭代法求解，结果如下:

```
A=
[[1 4 9 1 4 7 2]
 [0 7 9 0 8 0 0]
 [0 2 6 7 3 0 2]
 [9 7 7 1 7 8 9]
 [9 7 0 0 2 6 5]
 [7 9 4 5 1 6 5]
 [8 4 5 0 7 5 5]]
Answer=
[[-2]
 [-5]
 [-9]
 [-2]
 [-6]
 [-8]
 [-8]]
b=
[[-201]
 [-164]
 [-112]
 [-296]
 [-153]
 [-199]
 [-203]]

Gauss-Seidel method:
GS method iterations =  1000
x =
[-1.17183147e+161  3.50565866e+160 -6.52238794e+160  1.25556967e+160
  4.27023510e+160  6.03565653e+160  1.04531789e+161]

SOR method:
SOR method iterations =  1000
x =
[-2.47854149e+207  8.63322181e+206 -1.42629145e+207  3.57468663e+206
  9.06250946e+206  1.32749998e+207  2.24654694e+207]
```

```
A=
[[4 6 3 9 0 2 4]
 [2 9 8 0 5 2 3]
 [4 6 2 9 1 8 5]
 [8 0 6 3 0 9 0]
 [1 2 9 2 4 7 9]
 [8 1 7 2 3 0 9]
 [9 7 4 4 2 8 1]]
Answer=
[[5]
 [1]
 [9]
 [1]
 [1]
 [1]
 [3]]
b=
[[ 76]
 [107]
 [ 77]
 [106]
 [128]
 [136]
 [105]]

Gauss-Seidel method:
D:\Python practise\practise\SZ\1.py:38: RuntimeWarning: overflow encountered in double_scalars
  x[i] = (b0[i] - np.dot(a0[i, :], x) + a0[i, i] * x[i]) / a0[i, i]
GS method iterations =  744
x =
[8.49592141e+306 7.06665091e+306             inf            -inf
             nan             nan             nan]

SOR method:
SOR method iterations =  609
x =
[6.27398383e+306 5.91912564e+306 1.82073482e+307 2.45804763e+305
            -inf             nan             nan]
```

结论

由以上结果可以看到，对严格对角占优矩阵应用 Gauss-Seidel 和 SOR 迭代法求解，它的解是收敛的，且由求解到相同精度时所需的迭代次数可以看出，与 Gauss-Seidel 迭代法相比，SOR 迭代法的收敛速度更快，稳定性更好。

对非严格对角占优矩阵应用 Gauss-Seidel 和 SOR 迭代法求解，它的解是发散的。

松弛因子 ω 对 SOR 算法结果的影响

对随机生成的严格对角占优矩阵使用 SOR 迭代法，设置松弛因子 ω 从 -2 到 4 递增，结果如下：

```
A=
[[37  9  9  8  3  2  6]
 [ 8 34  9  8  2  1  6]
 [ 7  3 20  2  2  6  0]
 [ 5  6  7 33  1  5  9]
 [ 5  6  6  9 34  7  1]
 [ 7  5  7  4  2 34  9]
 [ 2  3  8  9  3  6 31]]
Answer=
[[-2]
 [-9]
 [-6]
 [ 4]
 [-3]
 [ 6]
 [-2]]
b=
[[-186]
 [-356]
 [-123]
 [  35]
 [-126]
 [  95]
 [ -78]]
SOR method iterations =
D:\Python practise\practise\SZ\1.py:62: RuntimeWarning: overflow encountered in double_scalars
  x[i] = (1 - omega)*x[i] - omega * (np.dot(a0[i, :], x) - b0[i] - a0[i, i] * x[i])/a0[i, i]
w =  -2  iterations =  298
x =
[1.93893778e+306 3.01886341e+306 4.35376414e+306 6.63907814e+306
             inf             inf             inf]

w =  -1.5  iterations =  366
x =
[4.00422265e+306 5.75112262e+306 7.78062141e+306             inf
             inf             inf             inf]

D:\Python practise\practise\SZ\1.py:62: RuntimeWarning: overflow encountered in multiply
  x[i] = (1 - omega)*x[i] - omega * (np.dot(a0[i, :], x) - b0[i] - a0[i, i] * x[i])/a0[i, i]
D:\Python practise\practise\SZ\1.py:62: RuntimeWarning: invalid value encountered in double_scalars
  x[i] = (1 - omega)*x[i] - omega * (np.dot(a0[i, :], x) - b0[i] - a0[i, i] * x[i])/a0[i, i]
w =  -1.0  iterations =  495
x =
[inf inf inf inf inf inf nan]

w =  -0.5  iterations =  861
x =
[2.53971066e+306 2.96438366e+306 3.38875977e+306 3.94107645e+306
 4.45267868e+306 4.47463782e+306             inf]

w =  0.0  iterations =  1000
```

```
w =  -0.5  iterations =  861
x =
[2.53971066e+306 2.96438366e+306 3.38875977e+306 3.94107645e+306
 4.45267868e+306 4.47463782e+306             inf]

w =  0.0  iterations =  1000
x =
[0. 0. 0. 0. 0. 0. 0.]

w =  0.5  iterations =  45
x =
[-2. -9. -6.  4. -3.  6. -2.]

w =  1.0  iterations =  17
x =
[-2. -9. -6.  4. -3.  6. -2.]

w =  1.5  iterations =  52
x =
[-2. -9. -6.  4. -3.  6. -2.]

w =  2.0  iterations =  1000
x =
[-2.95315273e+28  2.04986692e+28 -3.43929511e+28  9.00401246e+28
 -2.06442590e+29  2.26753767e+28 -2.68697552e+28]

w =  2.5  iterations =  1000
x =
[ 1.23400868e+225 -2.92659016e+225 -4.91390721e+225  1.02250748e+226
 -4.61120780e+225 -3.84457865e+225 -1.07570350e+225]

w =  3.0  iterations =  832
x =
[-1.08237378e+306  4.70198155e+305 -1.38291176e+306  5.55600942e+306
             -inf              inf              nan]

w =  3.5  iterations =  637
x =
[ 7.42484123e+305  1.02675798e+306  2.93978471e+306 -1.17017504e+307
              nan              nan              nan]

w =  4.0  iterations =  533
x =
[-5.20374610e+306  7.25738840e+306  5.36935234e+306             -inf
              nan              nan              nan]
```

通过观察可以看出，当 $\omega < 0$ 或 $\omega > 2$ 时，SOR 迭代法结果是发散的，当 $\omega \in [0, 2]$ 时，SOR 迭代法的结果是收敛的，且 ω 越接近 1，收敛速度越快。

参考

https://blog.csdn.net/zry1318/article/details/85065979