实验四

张天乐 计96 2018011038

上机题2

实验内容

分别用雅可比, G-S 和 SOR 方法求线性方程组 $Ay = b$ 的解

(1) 对 $\varepsilon = 1$, $a = 1/2$, $n = 1000$ ， 分别用雅可比, G-S 和 SOR 方法求上述线性方程组的解, 然后比较与精确解的误差

生成矩阵 A 和向量 b

In [ ]:
```
import numpy as np

eps = 1
a = 0.5
n = 1000
h = 1 / n

def generate_A(eps, n):
    h = 1 / n
    A = np.zeros((n - 1, n - 1))
    for i in range(n - 1):
        if i != 0:
            A[i][i-1] = eps - h / 2
        A[i][i] = -2 * eps
        if i != n - 2:
            A[i][i+1] = eps + h / 2
    return A

def generate_b(eps, n):
    h = 1 / n
    b = np.ones((n - 1)) * a * h * h
    b[-1] -= eps + h / 2
    return b

A = generate_A(eps, n)
b = generate_b(eps, n)
```

雅可比方法

In [ ]:
```
def Jacobi(A, b):
    n = np.shape(b)[0]
    x = np.ones_like(b)
    step = 0
    while True:
        y = np.copy(x)
        for i in range(n):
```

```
                x[i] = b[i]
                if i > 0:
                    x[i] -= A[i][i - 1] * y[i - 1]
                if i < n - 1:
                    x[i] -= A[i][i + 1] * y[i + 1]
                x[i] /= A[i][i]
            step += 1
            if np.max(abs(x - y)) < 1e-5:
                break
        print('Jacobi 迭代步数 {}'.format(step))
        return x

y_jacobi = Jacobi(A, b)
```

```
Jacobi 迭代步数 44208
```

G-S 方法

In [ ]:

```
def GS(A, b):
    n = np.shape(b)[0]
    x = np.ones_like(b)
    step = 0
    while True:
        y = np.copy(x)
        for i in range(n):
            x[i] = b[i]
            if i > 0:
                x[i] -= A[i][i - 1] * x[i - 1]
            if i < n - 1:
                x[i] -= A[i][i + 1] * x[i + 1]
            x[i] /= A[i][i]
        step += 1
        if np.max(abs(x - y)) < 1e-5:
            break
    print('G-S 迭代步数 {}'.format(step))
    return x

y_GS = GS(A, b)
```

```
G-S 迭代步数 22380
```

SOR 方法（松弛因子 $\omega = 1.9$ ）

In [ ]:

```
def SOR(A, b, w):
    n = np.shape(b)[0]
    x = np.ones_like(b)
    step = 0
    while True:
        y = np.copy(x)
        for i in range(n):
            x_gs = b[i]
            if i > 0:
                x_gs -= A[i][i - 1] * x[i - 1]
            if i < n - 1:
                x_gs -= A[i][i + 1] * x[i + 1]
            x_gs /= A[i][i]
            x[i] = (1 - w) * x[i] + w * x_gs
        step += 1
        if np.max(abs(x - y)) < 1e-5:
            break
```

```
    print('SOR 迭代步数 {}'.format(step))
    return x

y_SOR = SOR(A, b, 1.9)
```

```
SOR 迭代步数 12113
```

计算精确解

In [ ]:
```
def fun_y(x, eps):
    return (1 - a) / (1 - np.exp(-1 / eps)) * (1 - np.exp(-x / eps)) + a * x

y_accurate = [fun_y(x, eps) for x in np.arange(h, 1, h)]
```

比较与精确解的误差

In [ ]:
```
def calc_error(y, y_acc):
    error = np.max(abs(y - y_acc))
    return error

print('Jacobi 误差:\t无穷范数 {}'.format(calc_error(y_jacobi, y_accurate)))
print('GS 误差:\t无穷范数 {}'.format(calc_error(y_GS, y_accurate)))
print('SOR 误差:\t无穷范数 {}'.format(calc_error(y_SOR, y_accurate)))
```

```
Jacobi 误差:	无穷范数 0.5040579727919645
GS 误差:	无穷范数 0.5007676580949794
SOR 误差:	无穷范数 0.05193297853120138
```

实验结论:

迭代步数:

- Jacobi : 44208
- G-S : 22380
- SOR ($\omega = 1.9$) : 12113

迭代速度：SOR > G-S > Jacobi

准确度：SOR > G-S > Jacobi

(2) 对 $\varepsilon = 0.1$, $\varepsilon = 0.01$, $\varepsilon = 0.001$ 考虑上述同样的问题。同时，观察变化 n 的值对解的准确度有何影响

测试不同的 ε 的影响

In [ ]:
```
def test(eps, n):
    print('\nn = {}, eps = {}'.format(n, eps))
    h = 1 / n

    A = generate_A(eps, n)
    b = generate_b(eps, n)

    y_jacobi = Jacobi(A, b)
    y_GS = GS(A, b)
    y_SOR = SOR(A, b, 1.3)
    y_accurate = [fun_y(x, eps) for x in np.arange(h, 1, h)]
```

```
    print('Jacobi 误差:\t无穷范数 {:.5f}'.format(calc_error(y_jacobi, y_accurate
    print('GS 误差:\t无穷范数 {:.5f}'.format(calc_error(y_GS, y_accurate)))
    print('SOR 误差:\t无穷范数 {:.5f}'.format(calc_error(y_SOR, y_accurate)))
```

In [ ]:
```
test(0.1, 1000)
test(0.01, 1000)
test(0.001, 1000)
```

```
n = 1000, eps = 0.1
Jacobi 迭代步数 23504
G-S 迭代步数 13971
SOR 迭代步数 17254
Jacobi 误差:	无穷范数 0.35335
GS 误差:	无穷范数 0.33114
SOR 误差:	无穷范数 0.20802

n = 1000, eps = 0.01
Jacobi 迭代步数 19695
G-S 迭代步数 11169
SOR 迭代步数 6564
Jacobi 误差:	无穷范数 0.02240
GS 误差:	无穷范数 0.00947
SOR 误差:	无穷范数 0.00464

n = 1000, eps = 0.001
Jacobi 迭代步数 2132
G-S 迭代步数 1571
SOR 迭代步数 1056
Jacobi 误差:	无穷范数 0.01752
GS 误差:	无穷范数 0.01739
SOR 误差:	无穷范数 0.01730
```

实验结论：

随着 ε 减小，三种迭代方法的收敛速度都变快了。Jacobi 方法的误差减小，G-S 和 SOR 方法的误差先减小后增大。

测试不同 n 的影响

In [ ]:
```
test(1, 100)
test(0.1, 100)
test(0.01, 100)
```

```
n = 100, eps = 1
Jacobi 迭代步数 7838
G-S 迭代步数 3960
SOR 迭代步数 2453
Jacobi 误差:    无穷范数 0.01049
GS 误差:        无穷范数 0.00987
SOR 误差:       无穷范数 0.00531

n = 100, eps = 0.1
Jacobi 迭代步数 2901
G-S 迭代步数 1664
SOR 迭代步数 1003
Jacobi 误差:    无穷范数 0.00564
GS 误差:        无穷范数 0.00300
SOR 误差:       无穷范数 0.00167

n = 100, eps = 0.01
Jacobi 迭代步数 269
G-S 迭代步数 186
SOR 迭代步数 114
Jacobi 误差:    无穷范数 0.01736
GS 误差:        无穷范数 0.01731
SOR 误差:       无穷范数 0.01728
```

三种迭代方法的误差（无穷范数）与 n, eps 的关系如下。发现 n 越大，准确度反而减小

n \ eps	**1**	**0.1**	**0.01**	**0.001**
100	0.01049, 0.00987, 0.00530	0.00564, 0.00300, 0.00166	0.01735, 0.01731, 0.01728	
1000	0.50405, 0.50076, 0.05193	0.35335, 0.33114, 0.20802	0.02240, 0.00947, 0.00464	0.01752, 0.01739, 0.01730