利用共轭梯度法求解大规模线性方程组

欧阳鑫健， 4121156012, 电信学部

3.2 用共轭梯度法求解线性方程组 $Ax = b$, 其中

$$A = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}, \quad b = \begin{pmatrix} -1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{pmatrix}.$$

矩阵 A 的阶数 n 分别取为 100, 200, 400, 指出计算结果是否可靠.

线性方程组 $Ax = b$

In [1]:

```
import numpy as np
from matplotlib import pyplot as plt

N = 100 # 系数矩阵A的大小
A = np.zeros((N,N))
b = np.zeros((N,1))

for i in range(N):
    A[i][i] = -2
    if i < N-1:
        A[i][i+1] = 1
    if i >0:
        A[i][i-1] = 1

b[0] = -1
b[N-1] = -1

# print('\n A: \n', A,'\n','b transpose: \n', b.T)
```

Attention

矩阵A为负定矩阵，故$-A$正定。 **求解$Ax = b$等价于求解$-Ax = -b$**

In [2]:

```
A = -A
b = -b
```

共轭梯度法求解

共轭梯度法的计算公式：

$$
\begin{cases}
d^{(0)} = r^{(0)} = b - Ax^{(0)} \\
\alpha_k = \dfrac{r^{(k)T} d^{(k)}}{d^{(k)T} A d^{(k)}} \\
x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)} \\
r^{(k+1)} = b - Ax^{(k+1)} \\
\beta_k = -\dfrac{r^{(k+1)T} A d^{(k)}}{d^{(k)T} A d^{(k)}} \\
d^{(k+1)} = r^{(k+1)} + \beta_k d^{(k)}
\end{cases}
$$

In [3]:

```
# define x0
x = np.ones((N,1))/2
# print('x0 transpose: \n', x.T)

d = np.zeros((N,1))
r = np.zeros((N,1))
# d0, r0
r = b-A@x
r0 = r
# print('r0 transpose: \n', r0.T)
d = r # d0 = r0
# print('d0 transpose: \n', d.T)
```

In [4]:

```
beta = 0
alpha = 0
R = np.zeros((N))

k = 0
while np.linalg.norm(r,2)/np.linalg.norm(r0,2) > 1e-6: #循环条件
    # 共轭梯度法的公式
    R[k] = np.linalg.norm(r,2)
    alpha = r.T @ d / (d.T @ A @ d)
    x = x + alpha * d
    #print('alpha',alpha, '\n')
    r = b - A @ x
    beta = - r.T @ A @ d / (d.T @ A @ d)
    #print('beta',beta, '\n')
    d = r + beta * d
    k = k + 1

print('共轭梯度法求得的方程的解：\n',x.T, '\n 迭代次数：', k)
```

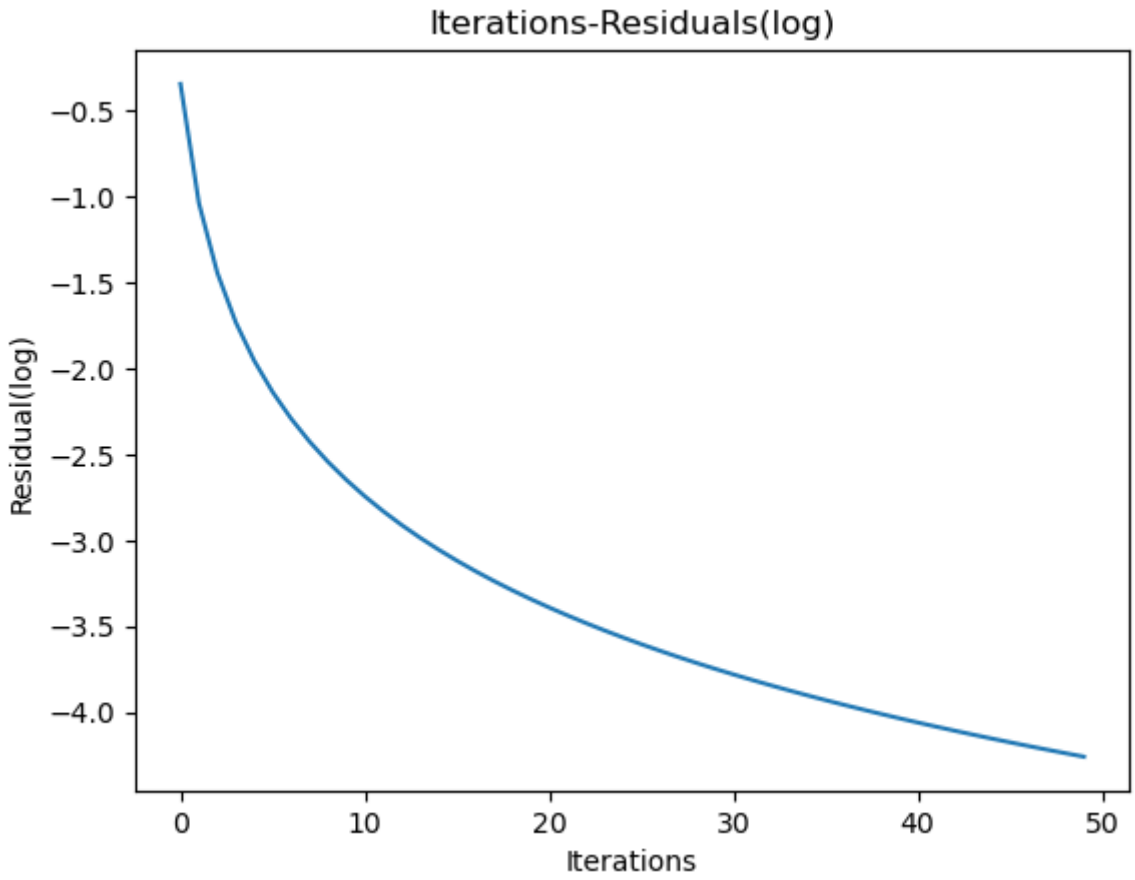
```
共轭梯度法求得的方程的解：
 [[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
  1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
  1. 1. 1. 1.]]
 迭代次数： 50
```

收敛速度图 ($N = 100$)

In [5]:

```
R = np.log(R)
#print(R)
plt.title("Iterations-Residuals(log)")
plt.xlabel("Iterations")
plt.ylabel("Residual(log)")
plt.plot(np.arange(len(R)),R)
plt.show()
```

```
<ipython-input-5-a28a3f4e96c7>:1: RuntimeWarning: divide by zero encou
ntered in log
  R = np.log(R)
```



可以看出，当 $N = 100$ 时，共轭梯度法在迭代50次后收敛，此时误差小于 10^{-6}。

当$N = 100, 200, 400$时，共轭梯度法的收敛情况

In [6]:

```
# 定义共轭梯度法的函数，参数为系数矩阵A的大小 N
def CG(N):
    A = np.zeros((N,N))
    b = np.zeros((N,1))

    for i in range(N):
        A[i][i] = -2
        if i < N-1:
            A[i][i+1] = 1
        if i >0:
            A[i][i-1] = 1
    b[0] = -1
    b[N-1] = -1

    # define x0
    x = np.ones((N,1))/2
    # d0, r0
    d = np.zeros((N,1))
    r = np.zeros((N,1))
    r = b-A@x
    r0 = r
    d = r # d0 = r0

    beta = 0
    alpha = 0
    R = np.zeros((N))
    k = 0
    while np.linalg.norm(r,2)/np.linalg.norm(r0,2) > 1e-6:
        R[k] = np.linalg.norm(r,2)
        alpha = r.T @ d / (d.T @ A @ d)
        x = x + alpha * d
        r = b - A @ x
        beta = - r.T @ A @ d / (d.T @ A @ d)
        d = r + beta * d
        k = k + 1

    return x, R, k #返回方程的解，误差序列和收敛次数
```

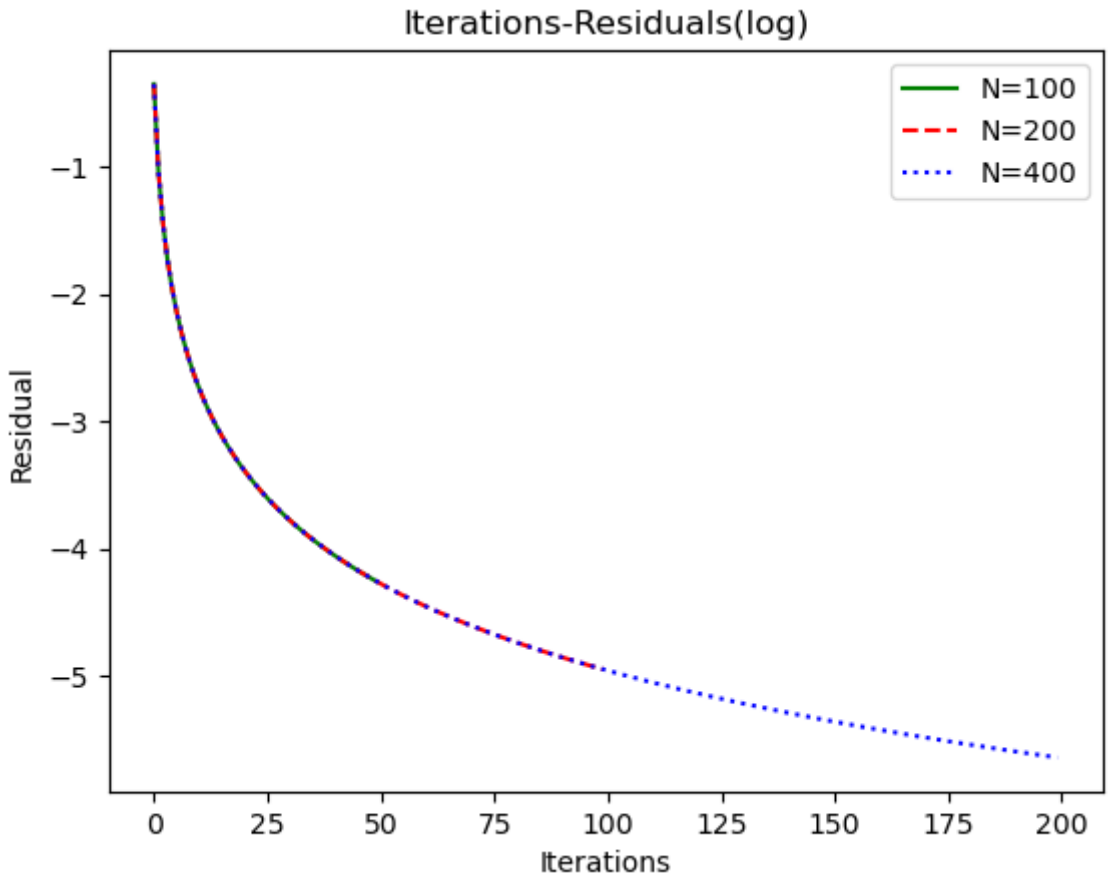
对于当$N = 100, 200, 400$时，绘制收敛速度图

In [7]:

```
x1, R1, k1 = CG(100)
x2, R2, k2 = CG(200)
x3, R3, k3 = CG(400)
#print(x1.T,'\n',x2.T,'\n',x3.T)
print('\n','N = 100, 迭代次数: ', k1, '\n','N = 200, 迭代次数: ', k2, '\n','N = 400, 迭

R1, R2, R3 = np.log(R1), np.log(R2), np.log(R3)
# print(R1,'\n',R2,'\n',R3)
plt.title("Iterations-Residuals(log)")
plt.xlabel("Iterations")
plt.ylabel("Residual")
plt.plot(np.arange(len(R1)), R1, linestyle = '-', color = 'green', label = 'N=100')
plt.plot(np.arange(len(R2)), R2, linestyle = 'dashed', color = 'red', label = 'N=200
plt.plot(np.arange(len(R3)), R3, linestyle = ':', color = 'blue', label = 'N=400')
plt.legend()
plt.show()
```

```
 N = 100, 迭代次数:  50
 N = 200, 迭代次数:  100
 N = 400, 迭代次数:  200
```

```
<ipython-input-7-c3e47f1fc278>:7: RuntimeWarning: divide by zero encou
ntered in log
  R1, R2, R3 = np.log(R1), np.log(R2), np.log(R3)
```



可以看出，当 $N = 100, 200, 400$ 时，共轭梯度法在迭代 $N/2$ 次后收敛，此时误差小于 10^{-6}。

最速下降法

取下降方向$p^(k)$为$\phi(x)$在x^k处的负梯度方向，计算可得

$$p^{(k)} = r^{(k)} = b - Ax^k = -\Delta\phi(x^{(k)})$$

In [8]:

```
N = 100
A = np.zeros((N,N))
b = np.zeros((N,1))

for i in range(N):
    A[i][i] = -2
    if i < N-1:
        A[i][i+1] = 1
    if i >0:
        A[i][i-1] = 1

b[0] = -1
b[N-1] = -1

# define x0
x = np.ones((N,1))/2
# print('x0 transpose: ','\n', x.T)

r = np.zeros((N,1))
# r0
r = b-A@x
# print('r0(p0) transpose', '\n', r.T)
```

In [9]:

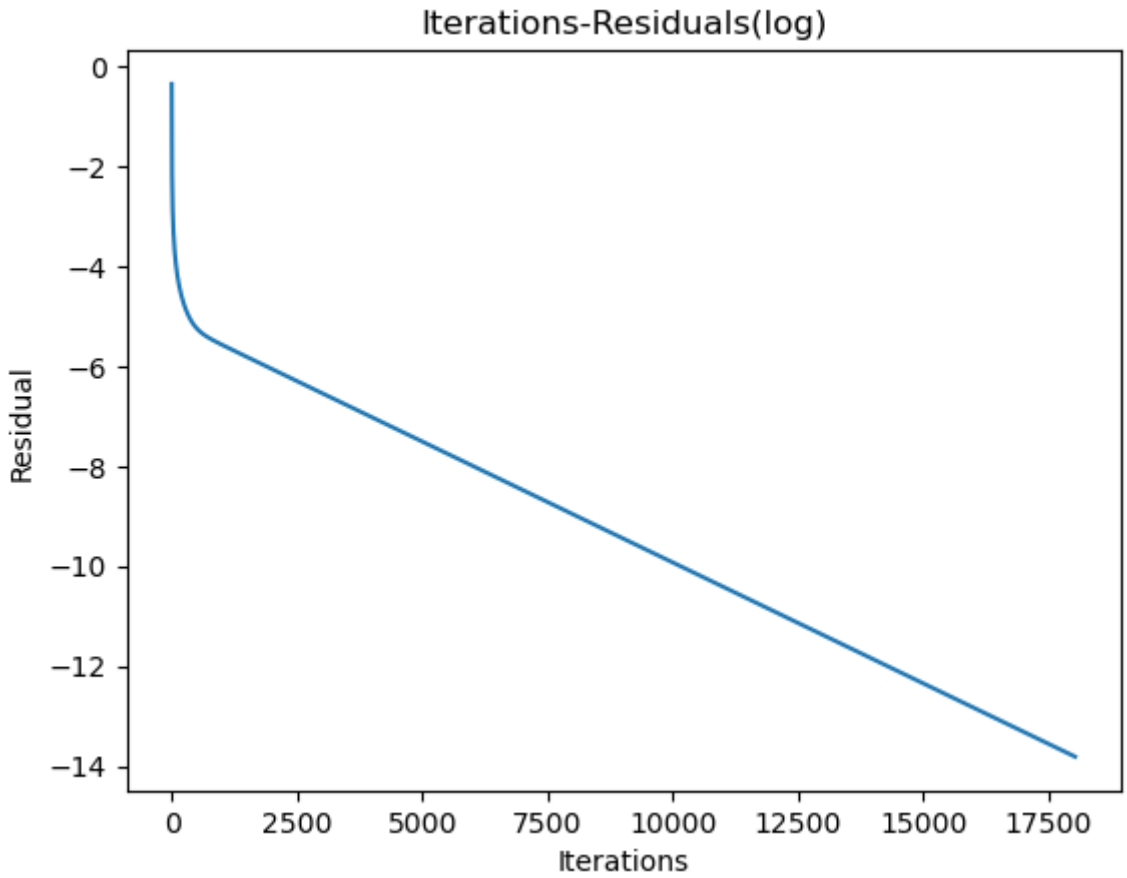
```
k = 0
R = []
while np.linalg.norm(r,2) > 1e-6:
    #最速下降法公式
    R.append(np.linalg.norm(r,2))
    alpha = r.T @ r / (r.T @ A @ r)
    x = x + alpha * r
    r = b - A @ x
    k = k + 1

print('最速下降法求得的方程组的解：  \n',x.T, '\n','迭代次数',k)
```

```
最速下降法求得的方程组的解：
 [[0.9999968  0.99999361 0.99999041 0.99998724 0.99998406 0.99998093
  0.99997777 0.99997468 0.99997157 0.99996854 0.99996548 0.99996252
  0.99995952 0.99995664 0.99995372 0.99995093 0.9999481  0.99994541
  0.99994268 0.99994009 0.99993748 0.99993502 0.99993252 0.99993019
  0.99992782 0.99992563 0.99992341 0.99992136 0.99991929 0.99991739
  0.99991548 0.99991374 0.999912   0.99991043 0.99990886 0.99990746
  0.99990608 0.99990485 0.99990365 0.99990261 0.9999016  0.99990075
  0.99989993 0.99989926 0.99989865 0.99989817 0.99989776 0.99989747
  0.99989727 0.99989717 0.99989717 0.99989727 0.99989747 0.99989776
  0.99989817 0.99989865 0.99989926 0.99989993 0.99990075 0.9999016
  0.99990261 0.99990365 0.99990485 0.99990608 0.99990746 0.99990886
  0.99991043 0.999912   0.99991374 0.99991548 0.99991739 0.99991929
  0.99992136 0.99992341 0.99992563 0.99992782 0.99993019 0.99993252
  0.99993502 0.99993748 0.99994009 0.99994268 0.99994541 0.9999481
  0.99995093 0.99995372 0.99995664 0.99995952 0.99996252 0.99996548
  0.99996854 0.99997157 0.99997468 0.99997777 0.99998093 0.99998406
  0.99998724 0.99999041 0.99999361 0.9999968 ]]
 迭代次数 18032
```

收敛速度图

In [10]:

```
R = np.log(R)
#print(R)
plt.title("Iterations-Residuals(log)")
plt.xlabel("Iterations")
plt.ylabel("Residual")
plt.plot(np.arange(len(R)),R)
plt.show()
```



可以看出，当 $N = 100$ 时，最速下降法迭代 18032 次后收敛，此时误差小于 10^{-6}。

共轭梯度法 v.s. 最速下降法

由计算结果和收敛速度图可知， 对于良态问题，共轭梯度法在**计算结果的准确度和收敛速度**上都优于最速下降法。