作业2

顾格非 3210103528

1. Implement CG algorithm to solve linear systems in which A is the Hilbert matrix, whose elements are $A(i,j)=\frac{1}{i+j-1}$. Set the right-hand-side to $b=(1,1,...,1)^T$ and the initial point to $x_0=0$. Try dimensions $n=5,8,12,20$ and show the performance of residual with respect to iteration numbers to reduce the residual below 10^{-6}.

首先我们准备好n阶的Hilbert matrix：

```
def hilbert_matrix(n):
    x, y_k = np.meshgrid(np.arange(1, n+1), np.arange(1, n+1))
    h = 1 / (x + y_k - 1)
    return h
```

然后是共轭梯度法的实现部分, 参照下面的步骤完成下面的代码:

$$
\begin{aligned}
\text{Set} r_0 &\leftarrow Ax_0-b, p_0 \leftarrow -r_0, k \leftarrow 0;\\
\alpha_k &\leftarrow \frac{r_k^T r_k}{p_k^T A p_k}\\
x_{k+1} &\leftarrow x_k+\alpha_k p_k\\
r_{k+1} &\leftarrow r_k+\alpha_k A p_k\\
\beta_{k+1} &\leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}\\
p_{k+1} &\leftarrow -r_{k+1}+\beta_{k+1} p_k\\
k &\leftarrow k+1
\end{aligned}
$$

```
def conjugate_gradient(A, b, x0, max_iter=1000, tol=1e-6):
    """
    使用共轭梯度法求解线性方程组 Ax=b
    :param A: 系数矩阵
    :param b: 右侧向量
    :param x0: 初始猜测解
    :param max_iter: 最大迭代次数
    :param tol: 收敛精度
    :return: 解向量
    """
    r0 = np.dot(A, x0) - b
    p0 = -r0
    x = x0
    i = 0
    while (i<max_iter):
        alpha = np.dot(r0.T, r0) / np.dot(p0.T, np.dot(A, p0))
        x = x + alpha * p0
        r1 = r0 + alpha * np.dot(A, p0)
        i+=1
        if np.linalg.norm(r1) < tol:
            print(np.linalg.norm(r1))
            break
        beta = np.dot(r1.T, r1) / np.dot(r0.T, r0)
        p0 = -r1 + beta * p0
        r0 = r1
    return x,i
```

使用不同的n进行实验，下面是不同n下 $residual \ \ norm < 10^{-6}$的迭代次数比较：

n	迭代次数	residual norm
5	6	2.624726839172203e-09
8	18	1.8915472281855566e-08
12	36	6.532510855464922e-07
20	74	6.77958998367044e-07

结果分析

- 可以看出，随着n的增大，共轭梯度法需要更多的迭代次数来达到所需的精度。
- 通过输出每次迭代的residential norm，可以看出n增大后迭代会呈现多次的波动，不过最后经过足够的迭代次数后都收敛到了所给的精度范围内。

---

2. Derive Preconditioned CG Algorithm by applying the standard CG method in the variables $\hat{x}$ and transforming back into the original variables x to see the expression of precondtioner M.

Preconditioned CG的主要目的是借用C 来改善原来的A矩阵的特征值分布，从而加速收敛，这里用了$\hat{x} = Cx$, 把其代入后要最小化的函数变成了:

$$\hat{\phi}(\hat{x}) = \frac{1}{2}\hat{x}^T(C^{-T}AC^{-1})\hat{x} - (C^{-T}b)^T\hat{x}$$

所以现在就是解$(C^{-T}AC^{-1})\hat{x} = C^{-T}b$这个方程：方法是把原来标准CG法中的A换成$C^{-T}AC^{-1}$, b 换成$C^{-T}b$计算:
在preconditioned CG中，我们令$p_0 = -y_0, My_0 = r_0$，于是得到下面两个方程:

$$\begin{aligned}
\hat{r}_0 = C^{-T}AC^{-1}\hat{x}_0 - C^{-T}b = C^{-T}\left(Ax_0 - b\right) = C^{-T}r_0 \\
\hat{p}_0 = -\hat{r}_0 = -C^{-T}r_0
\end{aligned}$$

$$\hat{\alpha}_0 = \frac{\left(C^{-T}r_0\right)^T\left(C^{-T}r_0\right)}{\left(C^{-1}\hat{p}_0\right)^T A\left(C^{-1}\hat{p}_0\right)} = \frac{r_0^T \cdot C^{-1}C^{-T}r_0}{\left(C^{-1}\hat{p}_0\right)^T A\left(C^{-1}\hat{p}_0\right)} = \frac{r_0^T y_0}{p_0^T A p_0}$$

由上面两式，可以推出M:

$$\begin{cases} C^{-1}C^{-T}r_0 = y_0 = M^{-1}r_0 \\ C^{-1}\hat{p}_0 = -C^{-1}C^{-T}r_0 = p_0 = -M^{-1}r_0 \end{cases} \Rightarrow M = C^TC$$

我们计算剩下的几步:

$$\hat{\beta}_{k+1} = \frac{\hat{r}_{k+1}^T\hat{r}_{k+1}}{\hat{r}_k^T\hat{r}_k} = \frac{r_{k+1}^T\left(C^{-1}C^{-T}r_{k+1}\right)}{r_k^T\left(C^{-1}C^{-T}r_k\right)} = \frac{r_{k+1}^T y_{k+1}}{r_k^T y_k}$$

$$\begin{aligned}
p_{k+1} = C^{-1}\hat{p}_{k+1} &= C^{-1}\left(-C^{-T}r_{k+1} + \hat{\beta}_{k+1}Cp_k\right) \\
&= -C^{-1}C^{-T}r_{k+1} + \hat{\beta}_{k+1}p_k \\
&= -y_{k+1} + \hat{\beta}_{k+1}p_k
\end{aligned}$$

于是我们可以不显示的在preconditioned CG中写出C，而是全换成用M表达，得到最终的preconditioned CG:

2 Preconditioned CG with simplifying

Given x_0, preconditioner M;
Set $r_0 = Ax_0 - b$;
Solve $My_0 = r_0$, $p_0 = -y_0, k = 0$;
while $r_k \neq 0$, **do**
  $\alpha_k = \frac{r_k^T y_k}{p_k^T A p_k}$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $r_{k+1} = r_k + \alpha_k A p_k$;
  Solve $My_{k+1} = r_{k+1}$;
  $\beta_{k+1} = \frac{r_{k+1}^T y_{k+1}}{r_k^T y_k}$;
  $p_{k+1} = -y_{k+1} + \beta_{k+1}p_k$;
  $k = k + 1$;
end while

3

3. Try to prove that when $\phi = \phi_k^c = \frac{1}{1-\mu_k}$ where $\mu_k = \frac{(s_k^T B_k s_k)(y_k^T H_k y_k)}{(s_k^T y_k)^2}$, the Broyden class

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} + \phi_k (s_k^T B_k s_k) v_k v_k^T$$

where

$$v_k = \left(\frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k} \right)$$

becomes sigular.

$$\begin{aligned}
\det\left(B_{k+1}\right) &= \det\left(B_{k+1}^{BFGS} + \phi_k\left(s_k^T B_k s_k\right) v_k v_k^T\right) \\
&= \det\left(B_{k+1}^{BFGS}\right) \det\left(I + \phi_k\left(s_k^T B_k s_k\right)\left(B_{k+1}^{BFGS}\right)^{-1} v_k v_k^T\right) \\
&= \det\left(B_{k+1}^{BFGS}\right)\left(1 + \phi_k\left(s_k^T B_k s_k\right) v_k^T \left(B_{k+1}^{BFGS}\right)^{-T} v_k\right) \\
&(\text{其中} B_{k+1}^{BFGS} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k})
\end{aligned}$$

$$\text{我们接下来证明} 1 + \phi_k\left(s_k^T B_k s_k\right) \cdot v_k^T \left(B_{k+1}^{BFGs}\right)^{-T} v_k = 0:$$

$$\begin{aligned}
&1 + \phi_k\left(s_k^T B_k s_k\right) \cdot v_k^T \left(B_{k+1}^{BFGs}\right)^{-T} v_k \\
&= 1 + \frac{\left(s_K^T y_k\right)^2}{\left(s_k^T y_k\right)^2 - \left(s_k^T B_k s_k\right)\left(y_k^T H_k g_k\right)} \cdot v_k^T \left[\left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T\right] v_k \\
&\text{这里用到了} (B_{k+1}^{BFGS})^{-1} = \left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s^T\right) + \rho_k s_k s^T \\
&= 1 + \frac{\left(s_K^T y_k\right)^2}{\left(s_k^T y_k\right)^2 - \left(s_k^T B_k s_k\right)\left(y_k^T H_k g_k\right)} \cdot \left(\frac{y_k^T}{y_k^T s_k} - \frac{s_k^T B_k}{s_k B_k s_k}\right) H_k \left(\frac{y_k}{y_k^T s_k} - \frac{B_k s_k}{s_k^T B_k s_k}\right) \\
&= 1 + \frac{\left(s_K^T y_k\right)^2}{\left(s_k^T y_k\right)^2 - \left(s_k^T B_k s_k\right)\left(y_k^T H_k g_k\right)} \cdot \left(s_k^T B_k s_k\right) \cdot \frac{\left(y_k^T H_k y_k\right)\left(s_k^T B_k s_k\right) - \left(y_k^T s_k\right)^2}{\left(y_k^T s_k\right)^2 \left(s_k^T B_k s_k\right)} \\
&= 1 - 1 = 0 \\
&\text{故} det(B_{k+1}) = 0\text{，即} B_{k+1} \text{是} singular \text{的.}
\end{aligned}$$

---

4. Using BFGS method to minimize the extended Rosenbrock function

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2],$$

with $x_0 = [-1.2, 1, \cdots, -1.2, 1]^T$, $x^* = [1, 1, \cdots, 1, 1]^T$ and $f(x^*) = 0$. Try different $n = 6, 8, 10$ and $\epsilon = 10^{-5}$. Moreover, using BFGS method to minimize the Powell singular function

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,$$

with $\epsilon = 10^{-5}$, $x_0 = [3, -1, 0, 1]^T$, $x^* = [0, 0, 0, 0]$ and $f(x^*) = 0$.

首先我们准备好计算$f(x)$，$grad(f(x))$的计算函数:

```python
def extended_rosenbrock_sympy(x):
    n = len(x)
    f = 0
    for i in range(n-1):
        f += 100 * (x[i+1] - x[i]**2)**2 + (1 - x[i])**2
    return f
def grad(x_k):
    df = [sympy.diff(extended_rosenbrock_sympy(x), xi).subs(zip(x, x_k)) for xi in x]
    df_func = sympy.lambdify(x, df)
    df_array = np.array(df_func(*x_k))
    return df_array
```

然后完成BFGS的主函数：

```
def BFGS(fun, grad, x0, max_iter=100, epsilon=1e-5):
    """
    fun: 目标函数
    grad: 目标函数的梯度
    x0: 初始点
    max_iter: 最大迭代次数
    epsilon: 收敛精度
    """
    n = len(x0)
    I = np.identity(n)
    H = I
    k = 0
    while (k<max_iter):
        # 计算梯度
        g = grad(x0)

        # 判断是否收敛
        if fun(x0) < epsilon:
            break
        # 更新步长
        p = -np.dot(H, g)

        # 一维搜索求最优步长alpha
        alpha = 1.0
        rho = 0.9
        c = 1e-4
        while fun(x0 + alpha * p) > fun(x0) + c * alpha * np.dot(g, p):
            alpha = rho * alpha

        # 更新x
        s = alpha * p
        x1 = x0 + s

        # 计算梯度差和参数差
        y_k = grad(x1) - g
        s_k = x1 - x0

        # 更新H(这里的H其实是inv(B_k))
        H = np.dot((I - np.outer(s_k, y_k) / np.dot(y_k, s_k)), np.dot(H, (I - np.outer(y_k, s_k) / np.dot(y_k,
s_k)))) + np.outer(s_k, s_k) / np.dot(y_k, s_k)

        # 更新x0
        x0 = x1
        k+=1
    # 返回最优解和最优值
    return k,x0, fun(x0)
```

$$B_{k+1}=B_k-\frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}+\frac{y_k y_k^T}{y_k^T s_k}$$

这里我不是按上面的公式直接更新B_{k+1}，而是选择更新$H_{k+1}=B_{k+1}^{-1}$，这样避免求B_{k+1}的逆带来的复杂。

$$H_{k+1}=\left(I_n-\frac{s_k y_k^T}{y_k^T s_k}\right)H_k\left(I_n-\frac{y_k s_k^T}{y_k^T s_k}\right)+\frac{s_k s_k^T}{y_k^T s_k}$$

结果：

n	迭代次数	误差
6	75	2.966154355880596e-06
8	83	1.4750701116694114e-06
10	113	1.840508350662231e-06

n=6时

$x=(0.99993813, 0.99997309, 1.00005015, 1.00015015, 1.00026079, 1.00045491)$

n=8 时 $x=(1.00000972, 0.99998852, 1.00001296, 0.99997958, 0.99993918, 0.99983456, 0.99959476, 0.99921299)$

n=10时

$$x = (1.00002206, 1.0000238, 0.99999981, 0.99998003, 1.00000715, 0.99998502, 0.99996121, 0.99982655, 0.99962816, 0.99928181)$$

4-2

minimize the Power singular function, 方法同上（就不帖代码了，可以在提交的文件夹中找到）

Result：

经过18次迭代，得到$x = (-0.0383479273638743, 0.00387012863609520, -0.0139938967644757, -0.0138523889653648)$,

此时的误差：4.85543892470445e-6 满足条件。