```python
import numpy as np
import scipy as sp
from numpy.random import randn
from scipy.linalg import orth
import time
import pandas as pd

'''
Step 1: Create X, Y and D, then compute A
'''
m = 1000
n = 100000
X = orth(randn(m, m))
Y = orth(randn(n, m))

r = 10
d = 4 * 10 ** (-3)
d1 = np.array([r - i + 1 for i in range(1, r + 1)]).reshape((r, 1))
d2 = np.full((m - r, 1), d)
d = np.vstack((d1, d2))
D = np.diag(d.reshape(m))

A = X.dot(D).dot(Y.T)

'''
Step 2: Compute A's svd, and record the time needed for svd
'''
A_svd_start_time = time.time()
U_A, D_A, V_A = sp.linalg.svd(a=A, full_matrices=False, lapack_driver="gesvd")
print("---SVD of A: %s seconds ---" % (time.time() - A_svd_start_time))

'''
Step 3: Get the top r left/right singulars vectors of U_A
'''
U_A_r = np.zeros((m, r))
V_A_r = np.zeros((r, n))
U_A_r = U_A[:, :r]
V_A_r = V_A[:r, :].T

'''
Step 4: Compute p for each col Ai/ each row Aj
'''
norm_A = np.linalg.norm(A)
norm_Ai = np.array([np.linalg.norm(A[:, i]) for i in range(n)])
pi = norm_Ai ** 2 / norm_A ** 2
norm_Aj = np.array([np.linalg.norm(A[j, :]) for j in range(m)])
pj = norm_Aj ** 2 / norm_A ** 2

sum_c = 0
Err_col = [[] for i in range(10)]
Err_row = [[] for i in range(10)]
```

```python
for iter in range(10):
    for c in range(r, r * 100):
        print("C = ", c)
        '''
        Step 5: Randomly choose c cols based on pi/ c rows based on pj
        '''
        #        c = 15

        cols = np.random.choice(n, c, p=pi)
        pi_c = pi[cols]
        B_col = A[:, cols] / np.sqrt(c * pi_c).reshape((1, c))
        #        B_col = B_col.dot(B_col.T)
        rows = np.random.choice(m, c, p=pj)
        pj_c = pj[rows]
        B_row = (A[rows, :] / np.sqrt(c * pj_c).reshape((c, 1))).T
        #        B_row = B_row.T.dot(B_row)

        '''
        Step 6: Compute B_col's top r left/B_row's top r right singular vectors
        '''
        U_B_col, D_B_col, V_B_col = sp.linalg.svd(a=B_col, full_matrices=False, lapack_driver="gesvd")
        U_r = U_B_col[:, :r]
        #        U_r = B_col.T.dot(U_B_col_r)/np.linalg.norm(B_col.T.dot(U_B_col_r),axis = 0)
        U_B_row, D_B_row, V_B_row = sp.linalg.svd(a=B_row, full_matrices=False, lapack_driver="gesvd")
        V_r = U_B_row[:, :r]
        #        V_r = B_row.T.dot(V_B_row_r)/np.linalg.norm(B_row.T.dot(V_B_row_r),axis = 0)

        '''
        Step 8: compute errors of U_B_col_r
                first: compute ||UrUr - UrUr|| ** 2 using power method
                second: take squre root
        '''
        # power method:
        b_L = np.random.rand(m)
        #        b_L_P = b_L
        for i in range(100):
            # calculate the matrix-by-vector product Ab
            #            b_L_P = b_L
            b_L1 = np.dot(U_r, np.dot(U_r.T, b_L)) - np.dot(U_A_r, np.dot(U_A_r.T, b_L))
            b_L2 = np.dot(U_r, np.dot(U_r.T, b_L1)) - np.dot(U_A_r, np.dot(U_A_r.T, b_L1))
            # calculate the norm
            b_L2_norm = np.linalg.norm(b_L2)
            # re normalize the vector
            b_L = b_L2 / b_L2_norm
        # use Rayleigh quotient ||UrUr - UrUr|| ** 2 =
        # (UrUr - UrUr)** 2 's largest eigenvalue
        b_L1 = np.dot(U_r, np.dot(U_r.T, b_L)) - np.dot(U_A_r, np.dot(U_A_r.T, b_L))
        b_L2 = np.dot(U_r, np.dot(U_r.T, b_L1)) - np.dot(U_A_r, np.dot(U_A_r.T, b_L1))
        norm_Err_col = np.dot(b_L.T, b_L2) / (np.dot(b_L.T, b_L))
        if norm_Err_col < 0: continue
        # UrUr - UrUr 's largest singular value =
        # the sqrt of (UrUr - UrUr)** 2 's eigenvalue
        norm_Err_col = np.sqrt(norm_Err_col)
```

```python
        # power method:
        b_R = np.random.rand(n)
        b_R_P = b_R
        for i in range(50):
            b_R_P = b_R
            # calculate the matrix-by-vector product Ab
            b_R1 = np.dot(V_r, np.dot(V_r.T, b_R)) - np.dot(V_A_r, np.dot(V_A_r.T, b_R))
            b_R2 = np.dot(V_r, np.dot(V_r.T, b_R1)) - np.dot(V_A_r, np.dot(V_A_r.T, b_R1))
            # calculate the norm
            b_R2_norm = np.linalg.norm(b_R2)
            # re normalize the vector
            b_R = b_R2 / b_R2_norm
        # use Rayleigh quotient ||UrUr - UrUr|| ** 2 =
        # (UrUr - UrUr)** 2 's largest eigenvalue
        b_R1 = np.dot(V_r, np.dot(V_r.T, b_R)) - np.dot(V_A_r, np.dot(V_A_r.T, b_R))
        b_R2 = np.dot(V_r, np.dot(V_r.T, b_R1)) - np.dot(V_A_r, np.dot(V_A_r.T, b_R1))
        norm_Err_row = np.dot(b_R.T, b_R2) / (np.dot(b_R.T, b_R))
        if norm_Err_row < 0: continue
        # UrUr - UrUr 's largest singular value =
        # the sqrt of (UrUr - UrUr)** 2 's eigenvalue
        norm_Err_row = np.sqrt(norm_Err_row)

        '''
        Step 9: compute relative errors of U_B_col_r/V_B_row_r
        '''
        relative_norm_Err_col = norm_Err_col
        Err_col[iter].append(relative_norm_Err_col)
        relative_norm_Err_row = norm_Err_row
        Err_row[iter].append(relative_norm_Err_row)

        print("relative_norm_Err_col = ", relative_norm_Err_col)
        print("relative_norm_Err_row = ", relative_norm_Err_row)

        if (relative_norm_Err_col <= 0.01) and (relative_norm_Err_row <= 0.01):
            break

    print("error <= 0.01 : c = ", c)
    sum_c = sum_c + c

avg_c = sum_c / 10
print("error <= 0.05 : avg_c = ", avg_c)

output_file = 'knn_output.txt'

error_col = pd.DataFrame(np.array(Err_col[0]))
error_row = pd.DataFrame(np.array(Err_row[0]))

error_col.to_csv("error_col.csv")
error_row.to_csv("error_row.csv")
```