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


def power_method_norm(U_r, U_A_r, m):
    b = np.random.rand(m)
    for i in range(10):  # set itertation counter of power method to 50
        # calculate the matrix-by-vector product Ab
        b1 = np.dot(U_r, np.dot(U_r.T, b)) - np.dot(U_A_r, np.dot(U_A_r.T, b))
        b2 = np.dot(U_r, np.dot(U_r.T, b1)) - np.dot(U_A_r, np.dot(U_A_r.T, b1))
        # calculate the norm
        b2_norm = np.linalg.norm(b2)
        # re normalize the vector
        b = b2 / b2_norm
    # use Rayleigh quotient ||UrUr - UrUr|| ** 2 =
    # (UrUr - UrUr)** 2 's largest eigenvalue
    b1 = np.dot(U_r, np.dot(U_r.T, b)) - np.dot(U_A_r, np.dot(U_A_r.T, b))
    b2 = np.dot(U_r, np.dot(U_r.T, b1)) - np.dot(U_A_r, np.dot(U_A_r.T, b1))
    norm_err = np.dot(b.T, b2) / (np.dot(b.T, b))
    # UrUr - UrUr 's largest singular value =
    # the sqrt of (UrUr - UrUr)** 2 's eigenvalue
    norm_err = np.sqrt(norm_err)
    return norm_err


def random_SVM_trials(trials, r, eps, m, n, X, Y):
    # 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

for trial in range(trials):
    for c in range(r, r * 50):
        '''
        Step 5: Randomly choose c cols based on pi/ c rows based on pj
        '''
        cols = np.random.choice(n, c, p=pi)
        pi_c = pi[cols]
        B_col = A[:, cols] / np.sqrt(c * pi_c).reshape((1, c))
        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

        '''
        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_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]

        '''
        Step 7: compute errors of U_B_col_r
                first: compute ||UrUr - UrUr|| ** 2 using power method
                second: take squre root
        '''
        # power method:
        norm_Err_col = power_method_norm(U_r, U_A_r, m)

        # power method:
        norm_Err_row = power_method_norm(V_r, V_A_r, n)

        '''
```

```
            Step 8: compute relative errors of U_B_col_r/V_B_row_r
            '''
            relative_norm_Err_col = norm_Err_col
            relative_norm_Err_row = norm_Err_row

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

        print(f"r = {r}, error <= {eps}, iter {trial} : c = {c}")
        sum_c = sum_c + c

    avg_c = sum_c / 10
    return avg_c


def main():
    '''
    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_list = [2, 5, 15, 20]
    avg_c_all = np.zeros(4)

    for i in range(4):
        avg_c = random_SVM_trials(10, r_list[i], 0.05, m, n, X, Y)
        print(f"When r = {r_list[i]} and eps = 0.05, average c = {avg_c}.")
        avg_c_all[i] = avg_c

    np.savetxt("avg_c_all.csv", avg_c_all, delimiter=",")


if __name__ == "__main__":
    main()
```