# Homework 5

Due: Monday, December 7, 2020

**Question 1.** Eigenvalues and Power Iterations:
(a) Generate a random, symmetric matrix **A** which is $m$ by $m$ where $m = 10$. Use the EIGS command in MATLAB (or the equivalent in Python) to give you the ground truth eigenvalues and eigenvectors.
(b) Find the largest eigenvalue with the power iteration method. Compare the accuracy of the method as a function of iterations.
(c) Find all ten eigenvalues by Rayleigh Quotient iteration and guessing initial "eigenvectors". Compare the accuracy of the method as a function of iterations and discuss your initial guesses to find all eigenvalue/eigenvector pairs.
(d) Repeat (b) and (c) with a random matrix that is not symmetric. Be sure to plot the eigenvalue in the complex plane.

(a) We can generate a random symmetric matrix using `toeplitz` function in MATLAB (diagonal values of Toeplitz matrices are equal, we can make them different by adding a random number). Please see the Example section for numerical outputs.

```
A=toeplitz(rand(m,1))+diag(rand(m,1));
[V,D]=eigs(A); % true eigenvectors and eigenvalues
```

(b) We construct the power iteration in the following way

```
v=rand(m,1); % randomly guess an initial eigenvector
v=v/norm(v);
for k=1:20
    w=A*v;
    v=w/norm(w);
    lambda=v'*A*v;
end
```

We define the accuracy of our method as

$$\text{Absolute error} = \left| \frac{\lambda^k - \lambda_1}{\lambda_1} \right|$$

where $\lambda^k$ is approximation at the $k$-th iteration and $\lambda_1$ is the largest eigenvalue. We plot the error as a function of iterations in Figure 1.

(c) In order to get all ten eigenvalues, we need to choose different initial eigenvectors such that the Rayleigh quotient is close to each eigenvalue. We will achieve this by doing the following: first do some trial iterations to get an approximated range of eigenvalues; draw initial eigenvectors from a random distribution that covers this range; do Rayleigh quotient iteration and check whether the eigenvalue obtained is different to all other eigenvalues already computed. If no, make a new guess and iterate again. We perform this loop until we find all ten eigenvalues.

```
I=eye(m);
lambda_vec=zeros(m,1);
v_vec=zeros(m,m);
j=1;
while sum(lambda_vec==0)>0 % loop until all eigenvalues are found
    v=-5+10*rand(m,1);
    v=v/norm(v);
    lambda=v'*A*v;
    for k=1:20
        w=(A-lambda*I)\v;
        v=w/norm(w);
        lambda=v'*A*v;
    end
    if all(round(lambda_vec-lambda,4)) % check if eigenvalue is new
        lambda_vec(j)=lambda;
        v_vec(:,j)=v;
        j=j+1;
    end
end
[¬,idx]=sort(lambda_vec,'descend'); % descending sort the eigenvalues
eig_values=lambda_vec(idx);
eig_vectors=v_vec(idx);
```

It is evident from Figure 1 that Rayleigh quotient iteration converges much faster than power iteration for both symmetric and non-symmetric matrices. As explained in Trefethen book, Rayleigh quotitent has a cubic converge, whereas power iteration converges linearly.

(d) We generate a non-symmetric matrix again using `toeplitz` function.

```
A=toeplitz(rand(m,1),rand(m,1))+diag(rand(m,1));
```

For non-symmetric matrices, since their eigenvectors are complex in general, we will modify our initial guess to include the imaginary parts (the rest of the code is the same as the symmetric case).

```
v=-5+10*rand(m,1)+(-5+10*rand(m,1))*1i;
```

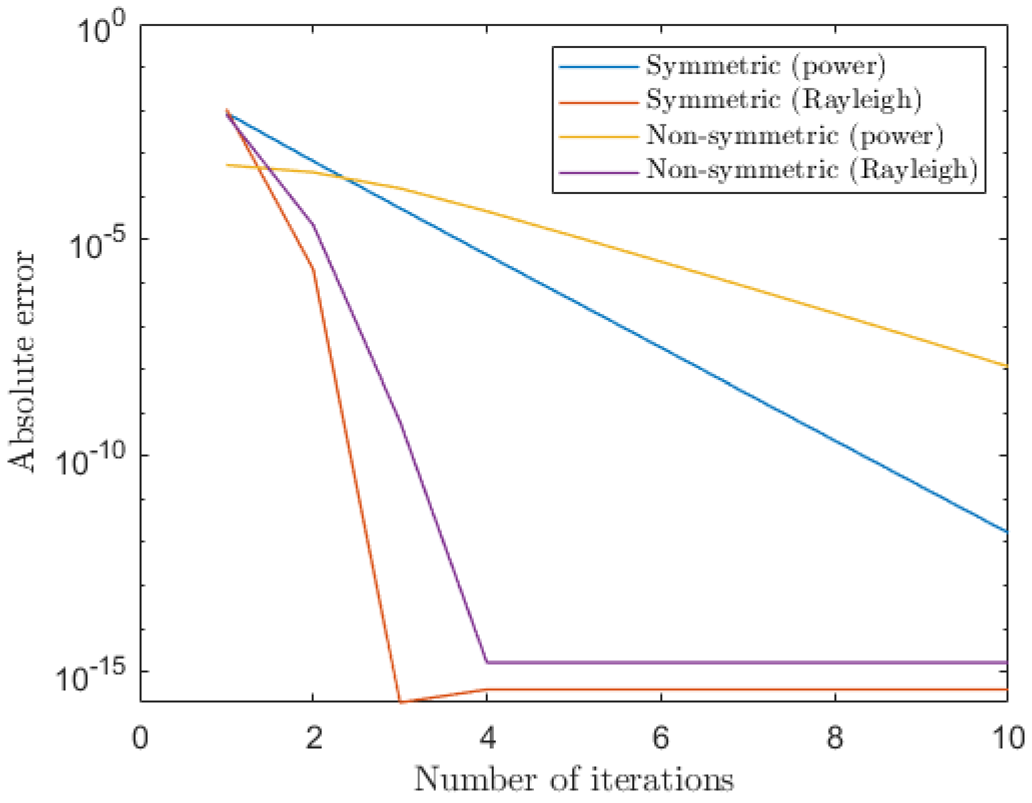

Figure 1: Absolute error of the largest eigenvalue as a function of iterations.

**Question 2.** Back to Yale Faces: Download the data set for CROPPED IMAGES.
(a) Power iterate on the matrix of images to find the dominant eigenvector and eigenvalue. Compare it to the leading order SVD mode.

We power iterate on the covariance matrix $\mathbf{A} = \mathbf{D}\mathbf{D}^{\mathbf{T}}$ and the dominant eigenvalue is `4.9100e+11`, which is equal to the square of the largest singular value, `7.0071e+05`, in agreement with theory. We also find that the dominant eigenvector is the same as the leading order SVD mode.

(b) Use randomized sampling to reproduce the SVD matrices: $\mathbf{U}, \mathbf{\Sigma}$ and $\mathbf{V}$.

Our approach follows closely the work by Kutz et al.[1], implemented as `rsvd` function in the Code section.

(c) Compare the randomized modes to the true modes along with the singular value decay as a function of the number of randomized samples.

[1]doi: 10.18637/jss.v089.i11

By computing the differences between the absolute values of randomized modes and true modes, we find that random sampling recovers the true modes very well (with machine precision).

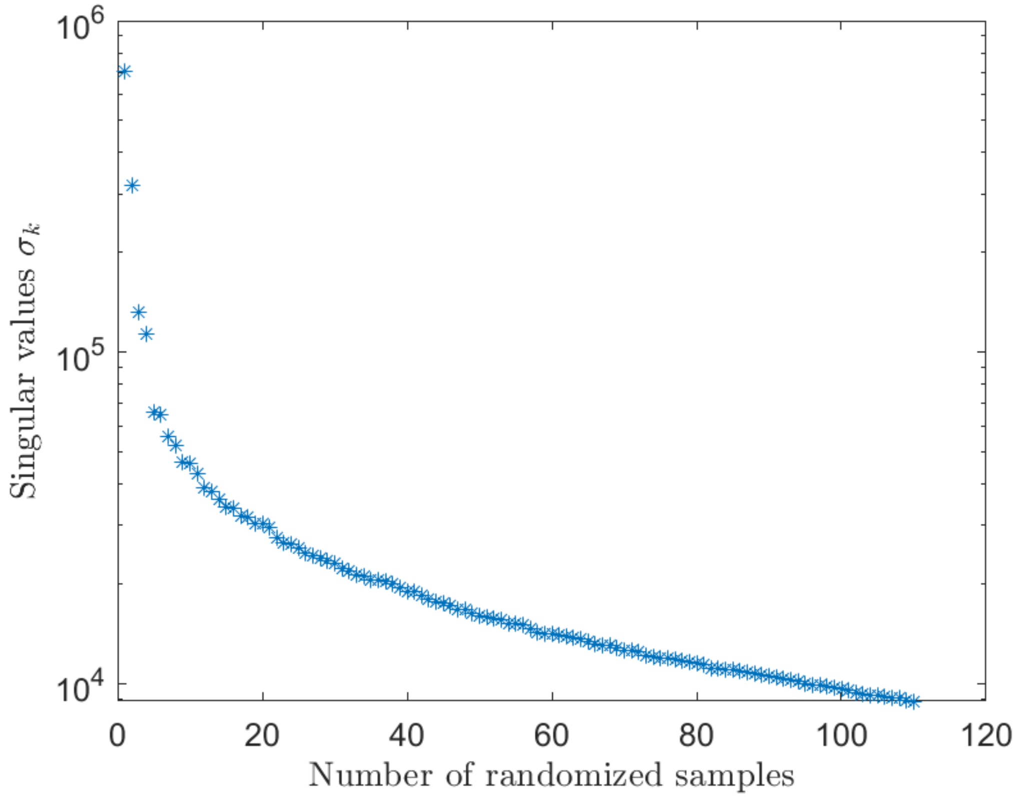

Figure 2: Singular value decay as a function of number of randomized samples.

## Example

```
>> A % symmetric

A =

    1.3093    0.5472    0.1386    0.1493    0.2575    0.8407    0.2543 ...
          0.8143    0.2435    0.9293
    0.5472    1.1559    0.5472    0.1386    0.1493    0.2575    0.8407 ...
          0.2543    0.8143    0.2435
    0.1386    0.5472    1.2104    0.5472    0.1386    0.1493    0.2575 ...
          0.8407    0.2543    0.8143
    0.1493    0.1386    0.5472    1.5753    0.5472    0.1386    0.1493 ...
          0.2575    0.8407    0.2543
    0.2575    0.1493    0.1386    0.5472    1.4326    0.5472    0.1386 ...
          0.1493    0.2575    0.8407
    0.8407    0.2575    0.1493    0.1386    0.5472    1.3110    0.5472 ...
          0.1386    0.1493    0.2575
    0.2543    0.8407    0.2575    0.1493    0.1386    0.5472    1.7901 ...
          0.5472    0.1386    0.1493
    0.8143    0.2543    0.8407    0.2575    0.1493    0.1386    0.5472 ...
          1.5446    0.5472    0.1386
    0.2435    0.8143    0.2543    0.8407    0.2575    0.1493    0.1386 ...
          0.5472    1.5090    0.5472
    0.9293    0.2435    0.8143    0.2543    0.8407    0.2575    0.1493 ...
          0.1386    0.5472    1.8765

>> D(1,1)

ans =

    5.0789

>> lambda

lambda =

    5.0789

>> [V D]=eig(A); diag(D)

ans =

   -0.5433
    0.3419
    0.5738
    0.7337
    1.2322
    1.2963
    1.6415
    2.0824
    2.2772
    5.0789

>> eig_values

eig_values =

    5.0789
    2.2772
    2.0824
    1.6415
    1.2963
    1.2322
    0.7337
    0.5738
    0.3419
   -0.5433

>> A % nonsymmetric

A =

    1.0312    0.3833    0.6173    0.5755    0.5301    0.2751    0.2486 ...
          0.4516    0.2277    0.8044
    0.7232    0.0750    0.3833    0.6173    0.5755    0.5301    0.2751 ...
          0.2486    0.4516    0.2277
    0.3474    0.7232    0.5807    0.3833    0.6173    0.5755    0.5301 ...
          0.2751    0.2486    0.4516
    0.6606    0.3474    0.7232    0.1321    0.3833    0.6173    0.5755 ...
          0.5301    0.2751    0.2486
    0.3839    0.6606    0.3474    0.7232    0.8471    0.3833    0.6173 ...
          0.5755    0.5301    0.2751
    0.6273    0.3839    0.6606    0.3474    0.7232    1.0342    0.3833 ...
          0.6173    0.5755    0.5301
    0.0216    0.6273    0.3839    0.6606    0.3474    0.7232    0.1120 ...
          0.3833    0.6173    0.5755
    0.9106    0.0216    0.6273    0.3839    0.6606    0.3474    0.7232 ...
          0.9844    0.3833    0.6173
    0.8006    0.9106    0.0216    0.6273    0.3839    0.6606    0.3474 ...
          0.7232    0.0632    0.3833
    0.7458    0.8006    0.9106    0.0216    0.6273    0.3839    0.6606 ...
          0.3474    0.7232    0.7289

>> D(1,1)

ans =

    5.0966

```

```
>> lambda

lambda =

    5.0966

>> [V D]=eig(A); diag(D)

ans =

   5.0966 + 0.0000i
  -0.7532 + 0.0000i
  -0.6712 + 0.0000i
  -0.2272 + 0.0000i
   0.2781 + 0.5388i
   0.2781 - 0.5388i
   0.5105 + 0.2148i
   0.5105 - 0.2148i
   0.2833 + 0.2555i
   0.2833 - 0.2555i

>> eig_values

eig_values =

   5.0966 + 0.0000i
  -0.7532 + 0.0000i
  -0.6712 + 0.0000i
   0.2781 - 0.5388i
   0.2781 + 0.5388i
   0.5105 - 0.2148i
   0.5105 + 0.2148i
   0.2833 + 0.2555i
   0.2833 - 0.2555i
  -0.2272 + 0.0000i
```

## Code

poweriter

```
clear; close all;
m=10;
A=toeplitz(rand(m,1)); % symmetric matrix
% A=toeplitz(rand(m,1),rand(m,1))+diag(rand(m,1)); % non−symmetric
[V,D]=eigs(A); % true eigenvectors and eigenvalues

% Power iteration
v=rand(m,1);
v=v/norm(v);
for k=1:20
    w=A*v;
    v=w/norm(w);
    lambda=v'*A*v;
    error(k)=100*abs((D(1,1)−lambda)/D(1,1));
end

% Rayleigh quotient iteration
I=eye(m);
lambda_vec=zeros(m,1);
v_vec=zeros(m,m);
j=1;
while sum(lambda_vec==0)>0 % loop until all eigenvalues are found
    v=−5+10*rand(m,1); % nonsymmetric ...
       v=−5+10*rand(m,1)+(−5+10*rand(m,1))*1i;
    v=v/norm(v);
    lambda=v'*A*v;
    for k=1:20
        w=(A−lambda*I)\v;
        v=w/norm(w);
        lambda=v'*A*v;
    end
    if all(round(lambda_vec−lambda,4)) % check if eigenvalue is new
        lambda_vec(j)=lambda;
        v_vec(:,j)=v;
        j=j+1;
    end
end
[¬,idx]=sort(lambda_vec,'descend'); % descending sort the eigenvalues
eig_values=lambda_vec(idx);
eig_vectors=v_vec(idx);
```

rsvd

```
function [U,S,V]=rsvd(A,k,p)
l=k+p; % slight oversampling
```

```
[¬,n]=size(A);
omega=randn(n,l); % random test matrix
Y=A*omega; % compute sketch
[Q,¬]=qr(Y); % form orthonormal basis
B=Q'*A; % project to low−dimensional space
[U0,S,V]=svd(B,'econ'); % econmic SVD
U=Q*U0; % recover singular vectors
```