Table of Contents


```
%{
This project explores power iteration and the effects of speedup and
accuracy it has on small scale matrices

For any questions or comments, please contact Ian Good
 (iangood@uw.edu)
Dec 5, 2020 - Matlab 2020b
%}

clear all;
close all;
clc;
```

Generating Eigenvalues for Random Symmetric and Non-Symmetric Matrices

```
m = 10;

A = 10*rand(m,m);
Asym = A + A'; %making it symmetric (diagonizable)
[eVecSym,eValSym] = eigs(Asym);
eValSymGT = diag(eValSym);
[eVec,eVal] = eigs(A);
eValGT = diag(eVal);
```

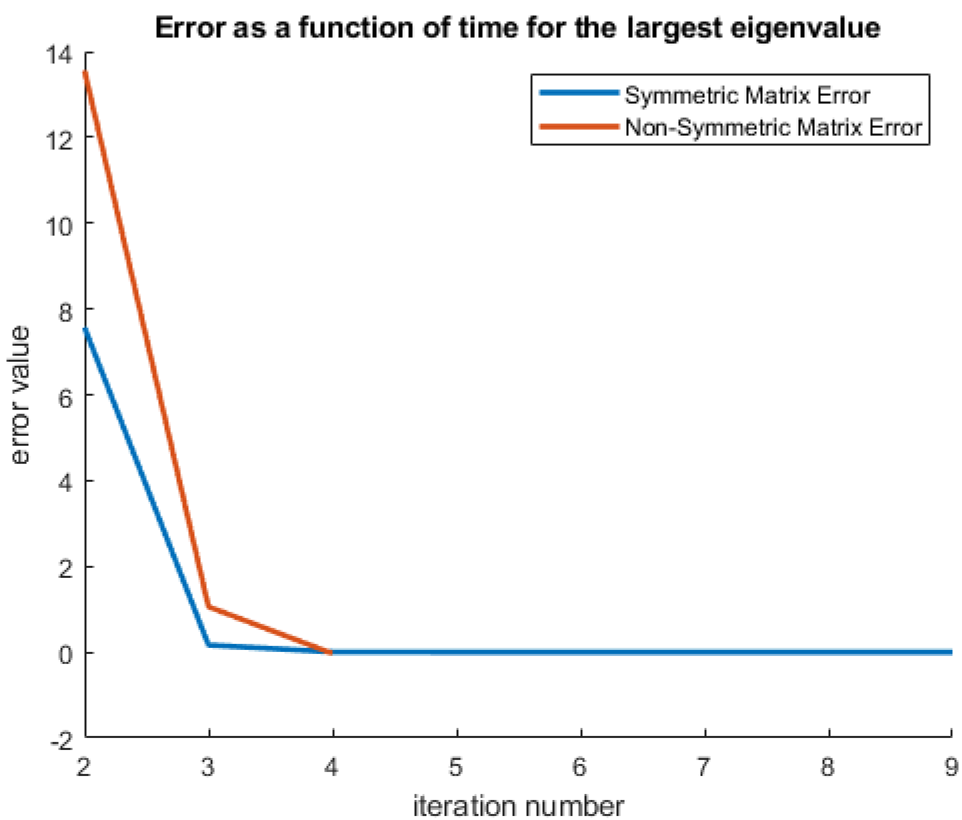
Power iteration method for finding the largest eigenvalue

```
vtSym = zeros(m,1);vtSym(1,1) = 1; %initial guess for eigenvectors
%(unit length)
errorSym = 1;
tolSym = 1e-9;
iter = 1;
    while (errorSym > tolSym)
        wSym = Asym*vtSym(:,iter);
        vtSym(:,iter+1) = wSym/norm(wSym,2);
        evalueSym = abs(vtSym(:,iter+1)'*Asym*vtSym(:,iter+1)); %we
 take
```

```
        %the absolute value here to prevent flip flopping between
 positve
        %and negative eigenvalues
        errorSym = abs(eValSymGT(1,1)) - abs(evalueSym);
        outSym(iter,:) = [ iter+1 errorSym]; %disp(outSym(iter,:));
        iter = iter+1;

    end
eVecPISym = vtSym(:,end);
eValPISym = evalueSym(:,end);
disp('The eigenvector for symmetric A using power iterations is:');
disp(num2str(eVecPISym));
disp(strcat('The eigenvalue for symmetric A using power iterations
 is:',num2str(eValPISym)));
 %Now calculate the answer from the non-symmetric A matrix
vt = zeros(m,1);vt(1,1) = 1; %initial guess for eigenvectors (unit
 length)
iter = 1;
error = 1;
tol = 1e-9;
    while(error > tol)
        w = A*vt(:,iter);
        vt(:,iter+1) = w/norm(w,2);
        evalue = abs(vt(:,iter+1)'*A*vt(:,iter+1));
        error = abs(eValGT(1,1)) - abs(evalue);
        out(iter,:) = [ iter+1 error]; %disp(out(iter,:));
        iter = iter+1;

    end
eVecPI = vt(:,end);
eValPI = evalueSym(:,end);
disp('The eigenvector for non-symmetric A using power iterations
 is:');
disp(num2str(eVecPI));
disp(strcat('The eigenvalue for non-symmetric A using power iterations
 is:',num2str(eValPI)));

%Plot the results
figure();
hold on; xlabel('iteration number'); ylabel('error value');
plot(outSym(:,1),outSym(:,2),'linewidth',[2]);
plot(out(:,1),out(:,2),'linewidth',[2]);
legend('Symmetric Matrix Error','Non-Symmetric Matrix Error')
title('Error as a function of time for the largest eigenvalue');

%{
Solutions
We can see the time to converge is linear for both symmetric and
non-symmetric matrices.
%}
```

The eigenvector for symmetric A using power iterations is:
0.34956
0.27235

```
0.37459
0.29772
0.29886
0.29613
0.32579
0.34102
0.29828
0.29371
The eigenvalue for symmetric A using power iterations is:101.2503
The eigenvector for non-symmetric A using power iterations is:
0.40915
0.19294
0.38376
0.33236
0.34853
0.32726
0.21866
0.34387
0.25678
 0.2776
The eigenvalue for non-symmetric A using power iterations is:101.2503
```



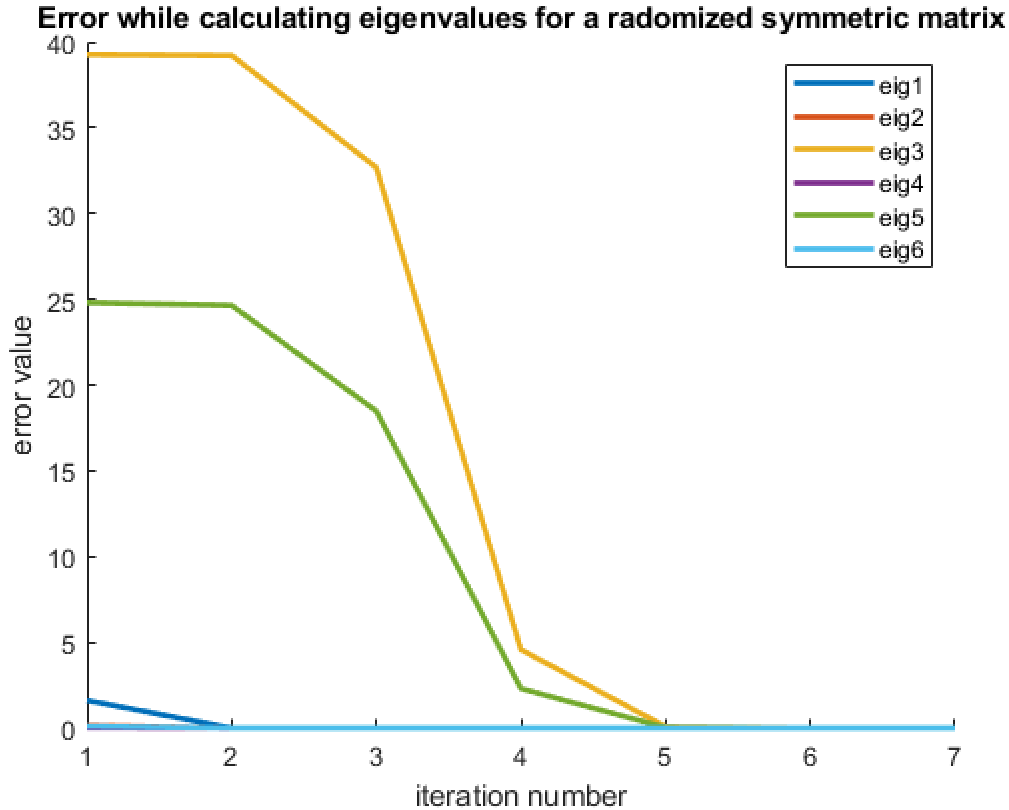
Finding all eigenvalues using Raleigh Quotient

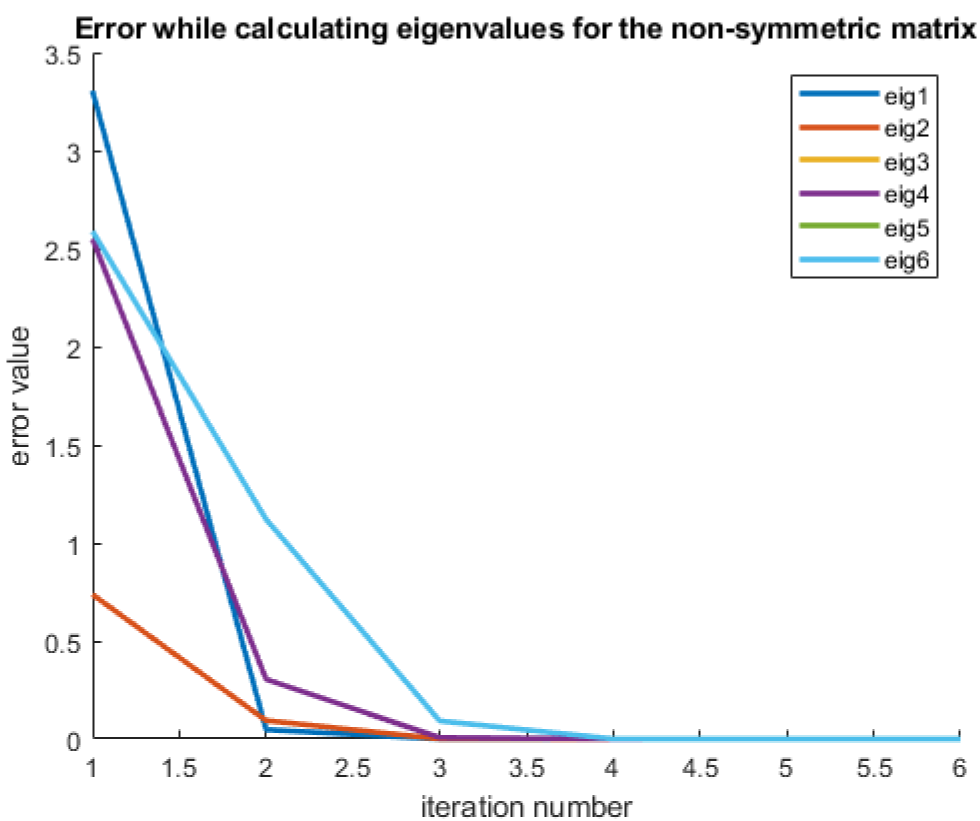
```
%Calculating values for the Symmetric Matrix
```

```
%Seed the solver with guesses that are close to the correct
 eigenvectors
% (closest decimal).
vtSymRQ = round(eVecSym,1);
vtSymRQ = vtSymRQ / norm(vtSymRQ,2); %normalize the starting
 vector(unit l)
errorSymRQ = ones(m,1);
tolSymRQ = 1e-9;
figure();
hold on; xlabel('iteration number'); ylabel('error value');
for jter = 1:length(vtSymRQ(1,:));
    evalueSymRQ(jter,1) = vtSymRQ(:,jter).'*Asym*vtSymRQ(:,jter);
    iter = 1;
    while errorSymRQ(jter,end) > tolSymRQ
        wSymRQ = inv((Asym-
evalueSymRQ(jter)*eye(length(Asym))))*vtSymRQ(:,jter);
        vtSymRQ(:,jter) = wSymRQ/norm(wSymRQ,2);
        evalueSymRQ(jter) =
 abs(vtSymRQ(:,jter)'*Asym*vtSymRQ(:,jter));
        errorSymRQ(jter,iter) = norm(Asym*vtSymRQ(:,jter)-
evalueSymRQ(jter)*vtSymRQ(:,jter));
        errorSymRQ(jter+1:end,iter) =1;
        iter = iter+1;
    end
    plot(errorSymRQ(jter,:),'linewidth',[2])
    hold on;
end
legend('eig1','eig2','eig3','eig4','eig5','eig6')
title('Error while calculating eigenvalues for a radomized symmetric
 matrix')


%Calculating values for the Non-Symmetric Matrix
%Seed the solver with guesses that are close to the correct
 eigenvectors
%(closest decimal)
vtRQ = round(eVec,1);
vtRQ = vtRQ / norm(vtRQ,2); %normalize the starting vector (unit
 length)
errorRQ = ones(m,1);
tolRQ = 1e-9;
figure();
hold on; xlabel('iteration number'); ylabel('error value');

for jter = 1:length(vtRQ(1,:))
    evalueRQ(jter,1) = vtRQ(:,jter).'*A*vtRQ(:,jter);
    iter = 1;
    while errorRQ(jter,end) > tolRQ
        wRQ = inv((A-evalueRQ(jter)*eye(length(A))))*vtRQ(:,jter);
        vtRQ(:,jter) = wRQ/norm(wRQ,2);
        evalueRQ(jter) = vtRQ(:,jter)'*A*vtRQ(:,jter);
        errorRQ(jter,iter) = norm(A*vtRQ(:,jter)-
evalueRQ(jter)*vtRQ(:,jter),2);
        errorRQ(jter+1:end,iter) =1;
```

```
        iter = iter+1;
    end
    plot(errorRQ(jter,:),'linewidth',[2])
    hold on;
end
legend('eig1','eig2','eig3','eig4','eig5','eig6')
title('Error while calculating eigenvalues for the non-symmetric
 matrix')
```

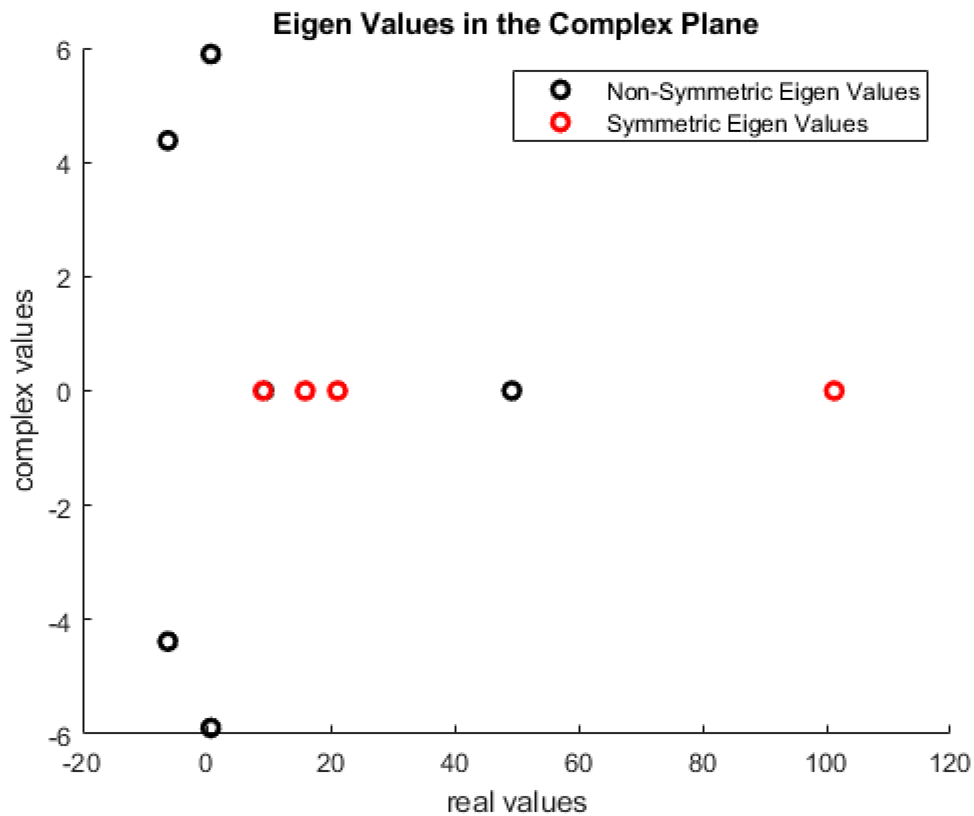
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 1.878993e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 2.839486e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 3.735477e-18.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 1.878993e-17.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 3.735477e-18.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 6.931148e-18.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 7.248864e-19.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 6.931148e-18.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 6.931148e-18.
Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 1.795636e-17.


Error while calculating eigenvalues for a radomized symmetric matrix
error value
iteration number
eig1
eig2
eig3
eig4
eig5
eig6
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
7


Error while calculating eigenvalues for the non-symmetric matrix
error value
iteration number
eig1
eig2
eig3
eig4
eig5
eig6
0
0.5
1
1.5
2
2.5
3
3.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6

Plotting the Eigenvalues

```
figure()
hold on; xlabel('real values'); ylabel('complex values');
eValComp = imag(evalueRQ); eValReal = real(evalueRQ);
plot(eValReal,eValComp,'ko','linewidth',[2])
plot(evalueSymRQ,zeros(1,length(evalueSymRQ)),'ro','linewidth',[2])
legend('Non-Symmetric Eigen Values','Symmetric Eigen Values')
title('Eigen Values in the Complex Plane')
```



Discussing the Results

```
%{
For the Rayleigh Quotient we see cubic convergence which is incredibly
computationally light (compared to linear convergence anyway). We do
 not
see a difference with non-symmetric matrices with the Rayleigh
 Quotient
either.


We seed the solver by using rounded values from the eigs command.
 While
initial guesses can be made in other ways, this method ensures all
eigenvalues will be discovered. There are many ways to properly seed a
```

```
power iteration, and as long as guesses are closest to a single
eigenvector, the nature of the methods used ensure the guess will be
 drawn
towards the exact value of the eigenvector.
With the Power Iteration Method (figure 1), we see linear convergence.
With the Rayleigh Quotient Method (figures 2 and 3), we see cubic
 convergence!!!

For additional methods for calculating eigenvalues and eigenvectors
 please
see here:
https://en.wikipedia.org/wiki/
Eigenvalue_algorithm#Iterative_algorithms
%}
```

Published with MATLAB® R2020b