Tutorial 5: The power method

You learned in class that you can compute the eigenvalues of a matrix **A** by multiplying an arbitrary vector repeatedly from the left by **A**. In this Tutorial you will learn how to implement this method in Matlab and how to extend it to compute more than one eigenvalue of a matrix.

The power method to find the dominant eigenvalue.

We apply the power method to the matrix

```
A = [2 -1  0
    -1  2 -1
     0 -1  2];
```

The method starts by selecting an arbitrary vector $\mathbf{x}_0$ whose largest entry is 1. You can generate an aribitrary vector using the randn command. You can learn how to use the randn command via the help function:

```
help randn
```

```
 RANDN Normally distributed pseudorandom numbers.
    R = RANDN(N) returns an N-by-N matrix containing pseudorandom values drawn
    from the standard normal distribution.  RANDN(M,N) or RANDN([M,N]) returns
    an M-by-N matrix. RANDN(M,N,P,...) or RANDN([M,N,P,...]) returns an
    M-by-N-by-P-by-... array. RANDN returns a scalar.  RANDN(SIZE(A)) returns
    an array the same size as A.

    Note: The size inputs M, N, P, ... should be nonnegative integers.
    Negative integers are treated as 0.

    R = RANDN(..., CLASSNAME) returns an array of normal values of the
    specified class. CLASSNAME can be 'double' or 'single'.

    R = RANDN(..., 'like', Y) returns an array of normal values of the
    same class as Y.

    The sequence of numbers produced by RANDN is determined by the settings of
    the uniform random number generator that underlies RAND, RANDN, and RANDI.
    RANDN uses one or more uniform random values to create each normal random
    value.  Control that shared random number generator using RNG.

    Examples:

       Example 1: Generate values from a normal distribution with mean 1
        and standard deviation 2.
          r = 1 + 2.*randn(100,1);

       Example 2: Generate values from a bivariate normal distribution with
       specified mean vector and covariance matrix.
          mu = [1 2];
          Sigma = [1 .5; .5 2]; R = chol(Sigma);
          z = repmat(mu,100,1) + randn(100,2)*R;

       Example 3: Reset the random number generator used by RAND, RANDI, and
       RANDN to its default startup settings, so that RANDN produces the same
       random numbers as if you restarted MATLAB.
          rng('default');
          randn(1,5)

       Example 4: Save the settings for the random number generator used by
```

```
    RAND, RANDI, and RANDN, generate 5 values from RANDN, restore the
    settings, and repeat those values.
       s = rng
       z1 = randn(1,5)
       rng(s);
       z2 = randn(1,5) % z2 contains exactly the same values as z1

    Example 5: Reinitialize the random number generator used by RAND,
    RANDI, and RANDN with a seed based on the current time.  RANDN will
    return different values each time you do this.  NOTE: It is usually
    not necessary to do this more than once per MATLAB session.
       rng('shuffle');
       randn(1,5)

 See Replace Discouraged Syntaxes of rand and randn to use RNG to replace
 RANDN with the 'seed' or 'state' inputs.

 See also RAND, RANDI, RNG, RANDSTREAM, RANDSTREAM/RANDN

 Reference page in Doc Center
    doc randn

 Other functions named randn

    codistributed/randn        distributed/randn
    codistributor1d/randn      gpuArray/randn
    codistributor2dbc/randn    RandStream/randn
```

We need an arbitrary vector of length 3 (because **A** is 3 x 3)

```
xo = randn(3,1)
```

```
xo = 3×1
    0.8015
   -1.5702
    1.3807
```

Next, we need to make sure that the largest entry of $\mathbf{x}_0$ is equal to one. One way of doing this is to find the maximum entry of $\mathbf{x}_0$ and divide each element of $\mathbf{x}_0$ by this maximum entry:

```
max_xo = max(abs(xo)); % find maximum entry in xo
xo = xo/max_xo;
xo
```

```
xo = 3×1
    0.5105
   -1.0000
    0.8793
```

We can now start the power method which requries that we obtain $\mathbf{x}_{k+1}$ from $\mathbf{x}_k$ in two steps:

1. Multiply $\mathbf{x}_k$ by **A** and define μ_k to be the largest entry of $\mathbf{A}\mathbf{x}_k$.
2. Set $\mathbf{x}_{k+1} = (1/\mu_k)\mathbf{A}\mathbf{x}_k$

This can be implemented with the following code:

```
x = xo; % initialize the method via xo
for kk = 1:100 % do 100 iterations
    x = A*x;
    m = max(abs(x));
    x = x*(1/m);
end
fprintf('Largest eigenvalue computed by the power method %g\n',m)
```

```
Largest eigenvalue computed by the power method 3.41421
```

You can check the result by using the eig command:

```
L = eig(A);
fprintf('Largest eigenvalue computed by eig %g\n',max(L))
```

```
Largest eigenvalue computed by eig 3.41421
```

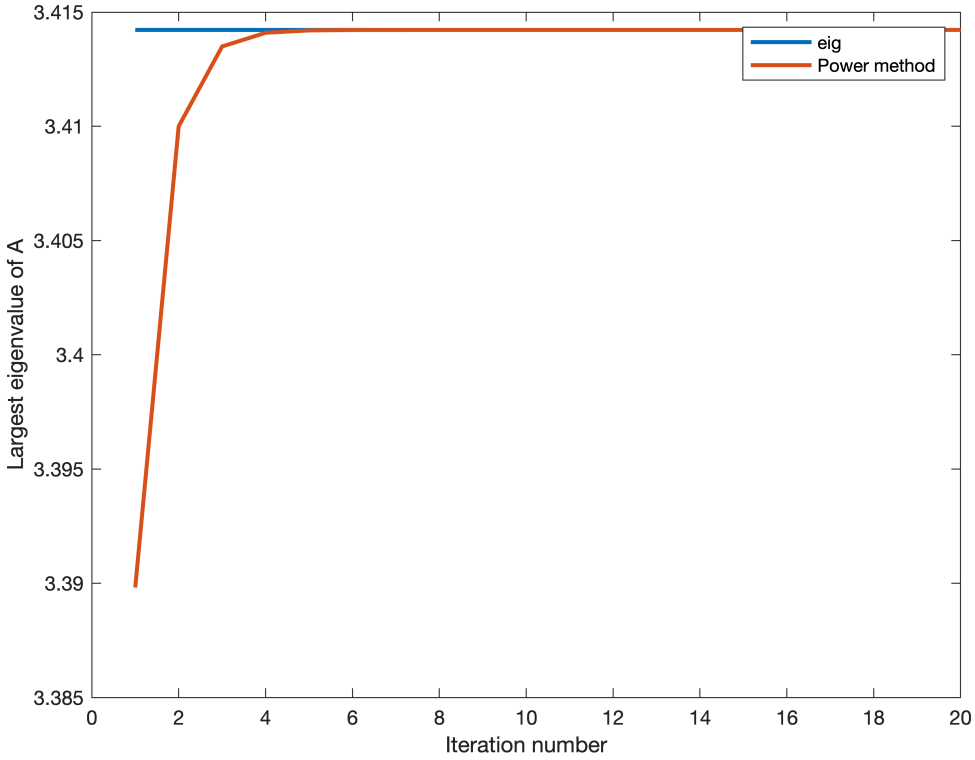
```
```

You can modify the code to save the value of μ_k and $\mathbf{x}_k$ during the iteration.

```
MuSave = zeros(20,1);
EvecSave = zeros(3,20);
x = xo; % initialize the method via xo
for kk = 1:20 % do 20 iterations
    x = A*x;
    m = max(abs(x));
    x = x*(1/m);
    EvecSave(:,kk) = x;
    MuSave(kk) = m;
end
```

You can then plot the results and see how quickly μ_k approaches the largest eigenvalue of **A**

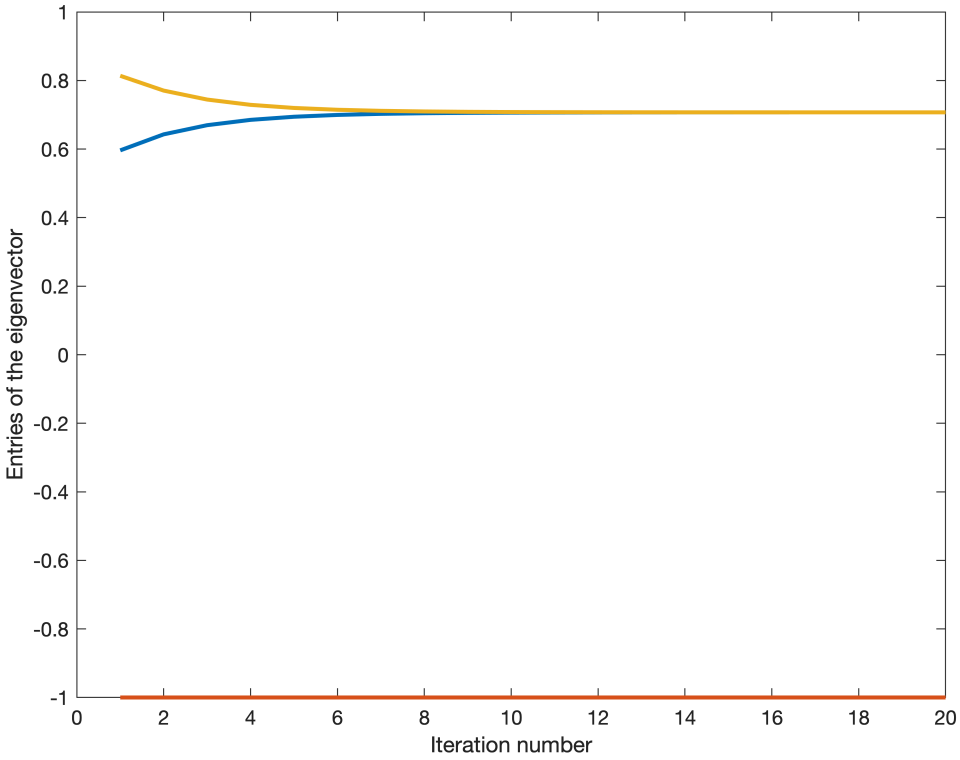
```
figure
plot([1 20],L(end)*[1 1],'LineWidth',2)
hold on, plot(MuSave,'-','LineWidth',2)
xlabel('Iteration number')
ylabel('Largest eigenvalue of A')
legend('eig',...
    'Power method')
```



After only 6 iterations, you already have a pretty accurate approximation of the largest eigenvalue of **A**.

You can also see how quickly the elements of $\mathbf{x}_k$ approach the elements of the eigenvector corresponding to the largest eigenvector of **A**

```
figure
plot(EvecSave(1,:),'LineWidth',2)
hold on, plot(EvecSave(2,:),'LineWidth',2)
hold on, plot(EvecSave(3,:),'LineWidth',2)
xlabel('Iteration number')
ylabel('Entries of the eigenvector')
```



Computing another eigenvalue of A.

The power method is useful for finding the largest eigenvalue of a matrix. But what if you are also interested in the second and third largest eigenvalue of **A**?

The idea is based in the following two facts:

1. If $\lambda \neq 0$ is an eigenvalye of an invertible matrix $\mathbf{A}$, then $1/\lambda$ is an eigenvalue of $\mathbf{A}^{-1}$
2. If λ is an eigenvalye of $\mathbf{A}$, then $\lambda - \alpha$ is an eigenvalue of $\mathbf{A} - \alpha\mathbf{I}$

Equipped with these two facts, it is easy to show that:

If λ is an eigenvalye of $\mathbf{A}$, then $1/(\lambda - \alpha)$ is an eigenvalue of $(\mathbf{A} - \alpha\mathbf{I})^{-1}$.

The proof is as follows. By Fact 2, it is clear that If λ is an eigenvalye of $\mathbf{A}$, then $(\lambda - \alpha)$ is an eigenvalue of $(\mathbf{A} - \alpha\mathbf{I})$. Combining this with Fact 1, proofs the claim.

The idea behind the "inverse power method" is to find eigenvalues of the matrix

$$\mathbf{B} = (\mathbf{A} - \alpha\mathbf{I})^{-1}$$

which, by the reasoning above are

$$\frac{1}{\lambda_1-\alpha},\frac{1}{\lambda_2-\alpha},\ldots,\frac{1}{\lambda_n-\alpha},$$

where $\lambda_1,\lambda_2,\ldots,\lambda_n$ are the eigenvalues of $\mathbf{A}$.Thus, if we set α to be approximately equal to λ_i, then the largest eigenvalue of $\mathbf{B}$ is

$$\mu_i=\frac{1}{\lambda_i-\alpha},$$

and the power method, applied to **B**, will find this eigenvalue. Given μ_i, we can then compute λ_i via:

$$\lambda_i=\alpha+\frac{1}{\mu_i}.$$

You can implement the inverse power method in Matlab as follows.

You decide on an alpha:

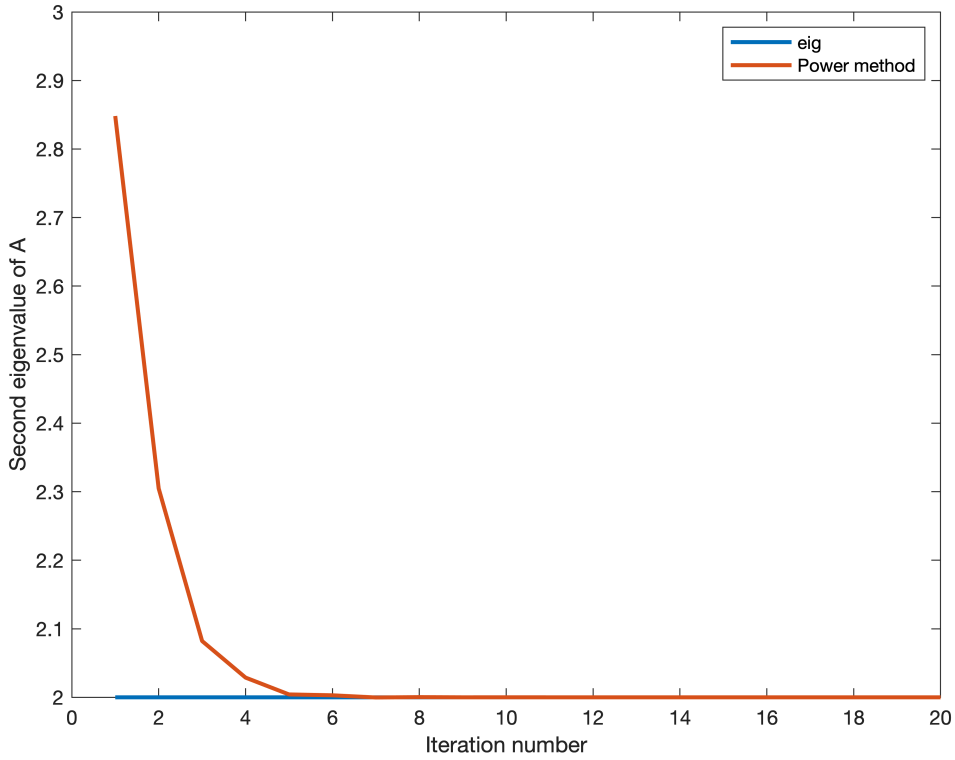
```
a = 1.5;
```

Then you apply the power method to the inverse of

```
B = (A-a*eye(3));
```

To search for eigenvalues near a.

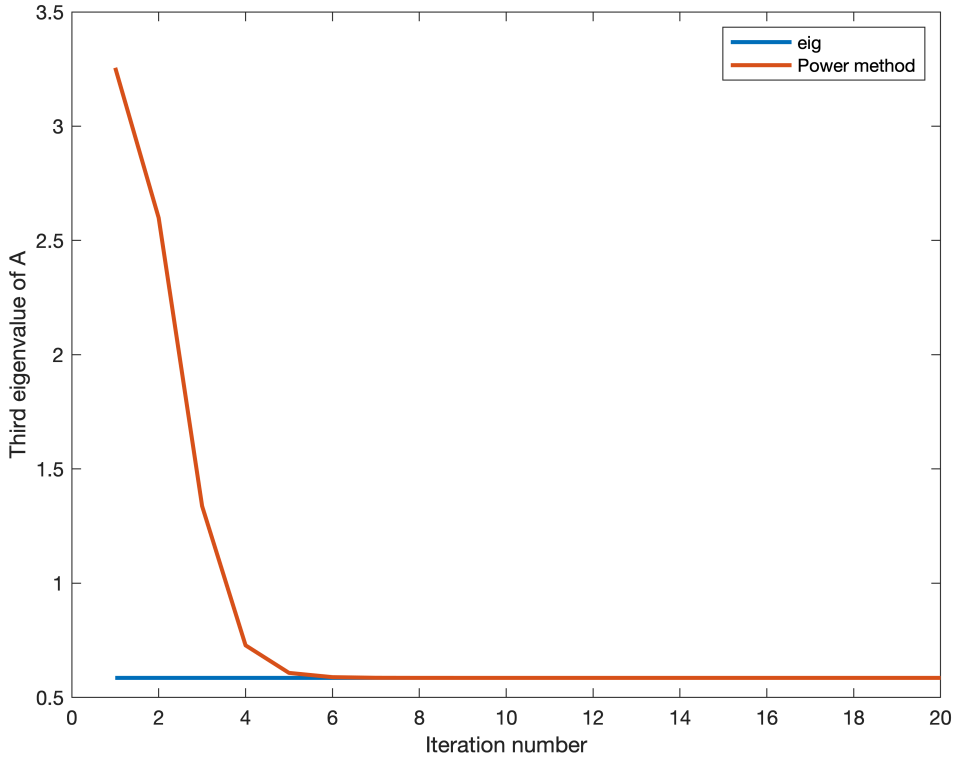
```
MuSave = zeros(20,1);
EvecSave = zeros(3,20);
x = xo; % initialize the method via xo
for kk = 1:20 % do 20 iterations
    x = B\x;
    m = max(abs(x));
    x = x*(1/m);
    EvecSave(:,kk) = x;
    MuSave(kk) = a+1/m;
end
figure
plot([1 20],L(end-1)*[1 1],'LineWidth',2)
hold on, plot(MuSave,'-','LineWidth',2)
xlabel('Iteration number')
ylabel('Second eigenvalue of A')
legend('eig','Power method')
```



You can see that the method works quite well for this example. We already found the first two eigenvalues which are, approximately, $\lambda_1 \approx 3.41$, $\lambda_2 \approx 2$. Assuming that the third eigenvalues is even smaller we try the inverse power method with a = 0.1.

```
a = .1;

B = (A-a*eye(3));
MuSave = zeros(20,1);
EvecSave = zeros(3,20);
x = xo; % initialize the method via xo
for kk = 1:20 % do 20 iterations
    x = B\x;
    m = max(abs(x));
    x = x*(1/m);
    EvecSave(:,kk) = x;
    MuSave(kk) = a+1/m;
end
figure
plot([1 20],L(1)*[1 1],'LineWidth',2)
hold on, plot(MuSave,'-','LineWidth',2)
xlabel('Iteration number')
ylabel('Third eigenvalue of A')
legend('eig','Power method')
```



Exercises

1. Proof the following statements:

    1. If $\lambda \neq 0$ is an eigenvalye of an invertible matrix $\mathbf{A}$, then $1/\lambda$ is an eigenvalue of $\mathbf{A}^{-1}$
    2. If λ is an eigenvalye of $\mathbf{A}$, then $\lambda - \alpha$ is an eigenvalue of $\mathbf{A} - \alpha\mathbf{I}$

Solution

Fact 1:

λ is an eigenvalye of $\mathbf{A}$. Thus

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

Multiply from the left by $\mathbf{A}^{-1}$

$$\mathbf{v} = \lambda\mathbf{A}^{-1}\mathbf{v}.$$

Multiply by $1/\lambda$:

$$\frac{1}{\lambda}\mathbf{v} = \mathbf{A}^{-1}\mathbf{v}.$$

Thus, $1/\lambda$ is an eigenvalue of $\mathbf{A}^{-1}$.

Fact 2:

λ is an eigenvalye of $\mathbf{A}$. Thus

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

Subtract $\alpha\mathbf{v}$ from both sides:

$$\mathbf{A}\mathbf{v} - \alpha\mathbf{v} = \lambda\mathbf{v} - \alpha\mathbf{v}.$$

Rearrange:

$$(\mathbf{A}\mathbf{v} - \alpha\mathbf{I})\mathbf{v} = (\lambda - \alpha)\mathbf{v}.$$

Thus, $\lambda - \alpha$ is an eigenvalue of $\mathbf{A} - \alpha\mathbf{I}$.

2. Use the power method and inverse power method to compute the two largest eigenvalues of the 100 x 100 matrix

```
clear % clear old variables
n = 100;
A = 2*eye(n,n)-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1);
```

Check your anwer with Matlab's eig command.

Solution

Power method for the largest eigenvalue of **A**

```
% starting vector
xo = randn(n,1);
max_xo = max(abs(xo)); % find maximum entry in xo
xo = xo/max_xo;

nIts = 100;
x = xo; % initialize the method via xo
for kk = 1:nIts
    x = A*x;
    m = max(abs(x));
    x = x*(1/m);
end
fprintf('Largest eigenvalue of A (1000 iterations): %g\n',m)
```

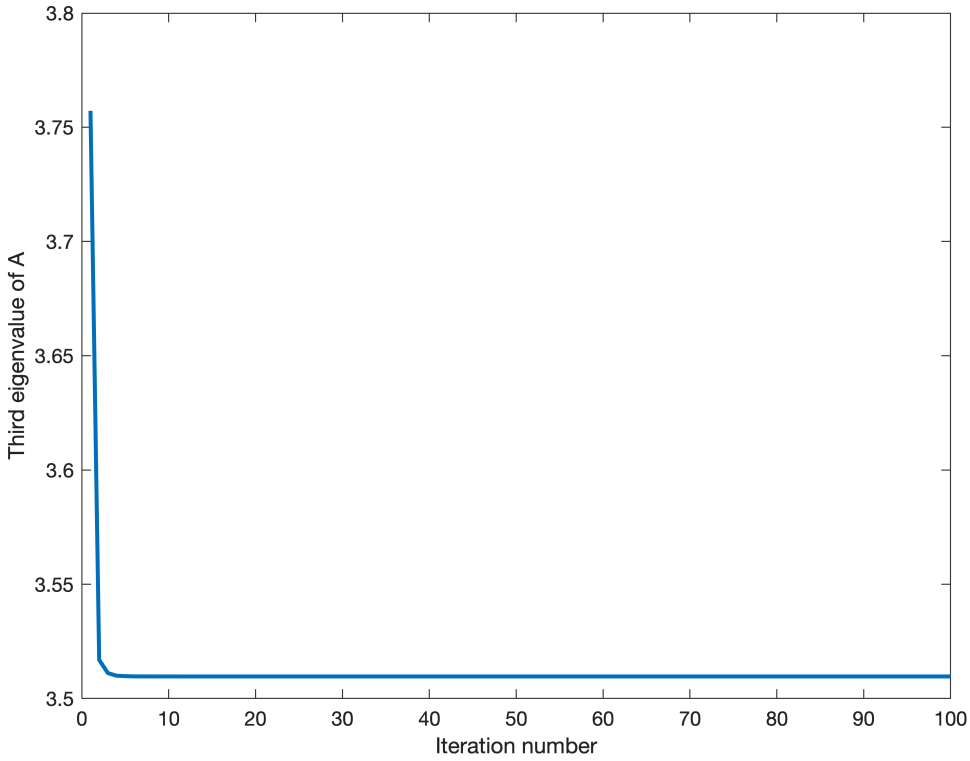
```
Largest eigenvalue of A (1000 iterations): 3.99093
```

Note that you need many iterations for the power method to converge.

Use the inverse power method to find the another eigenvalue:

```
a = 3.5;
```

```
B = (A-a*eye(n));
MuSave = zeros(nIts,1);
x = xo; % initialize the method via xo
for kk = 1:nIts % do 100 iterations
    x = B\x;
    m = max(abs(x));
    x = x*(1/m);
    MuSave(kk) = a+1/m;
end
figure
plot(MuSave,'-','LineWidth',2)
xlabel('Iteration number')
ylabel('Third eigenvalue of A')
```



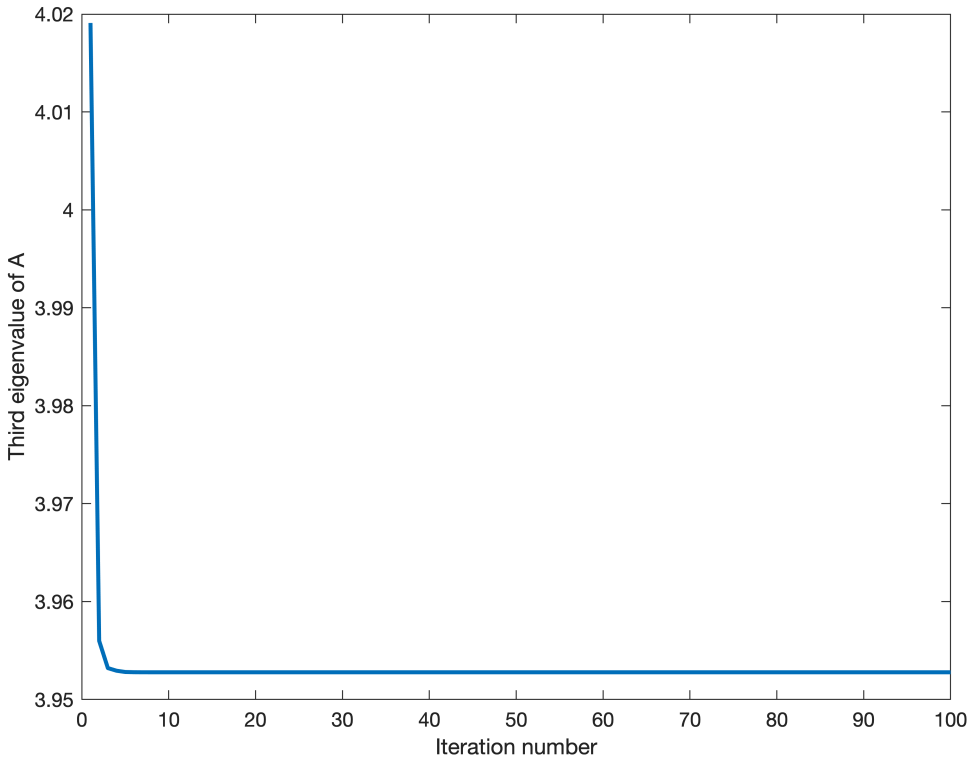

```
fprintf('Another eigenvalue of A (1000 iterations): %g\n',MuSave(end))
```

```
Another eigenvalue of A (1000 iterations): 3.50965
```

There is also another eigenvalues near the largest eigenvalue

```
a = 3.95;
B = (A-a*eye(n));
MuSave = zeros(nIts,1);
x = xo; % initialize the method via xo
for kk = 1:nIts % do 100 iterations
    x = B\x;
    m = max(abs(x));
```

```
    x = x*(1/m);
    MuSave(kk) = a+1/m;
end
figure
plot(MuSave,'-','LineWidth',2)
xlabel('Iteration number')
ylabel('Third eigenvalue of A')
```




```
fprintf('Another eigenvalue of A (1000 iterations): %g\n',MuSave(end))
```

```
Another eigenvalue of A (1000 iterations): 3.95278
```

In this example, using the power method is difficult because many eigenvalues are very near each other. This makes the iteration slow and it makes it difficult to come up with good guesses a to start the inverse power iteration.

You can use matlab's eig command to find the eigenvalues of A:

```
[U,L] = eig(A);
evals = diag(L); % eigenvalues are on the diagonal of L
evals = sort(evals,'descend'); % sort eigenvalues largest to smallest
disp('The five largest eigenvalues of A are:')
```

```
The five largest eigenvalues of A are:
```

```
evals(1:5)
```

```
ans = 5×1
    3.9990
    3.9961
    3.9913
    3.9845
    3.9759
```

As you can see, the large eigenvalues are quite near each other.