Tutorial 4: Eigenvalues and diagonalization

In this Tutorial you will learn how to compute eigenvalues and eigenvectors in Matlab and how to diagonalize a symmetric matrix.

Computing eigenvalues and eigenvectors

You can use eig to compute the eigenvalues and eigenvectors of a matrix. Use help to find out about the eig command.

```
help eig
```

```
 EIG    Eigenvalues and eigenvectors.
    E = EIG(A) produces a column vector E containing the eigenvalues of
    a square matrix A.

    [V,D] = EIG(A) produces a diagonal matrix D of eigenvalues and
    a full matrix V whose columns are the corresponding eigenvectors
    so that A*V = V*D.

    [V,D,W] = EIG(A) also produces a full matrix W whose columns are the
    corresponding left eigenvectors so that W'*A = D*W'.

    [...] = EIG(A,'nobalance') performs the computation with balancing
    disabled, which sometimes gives more accurate results for certain
    problems with unusual scaling. If A is symmetric, EIG(A,'nobalance')
    is ignored since A is already balanced.

    [...] = EIG(A,'balance') is the same as EIG(A).

    E = EIG(A,B) produces a column vector E containing the generalized
    eigenvalues of square matrices A and B.

    [V,D] = EIG(A,B) produces a diagonal matrix D of generalized
    eigenvalues and a full matrix V whose columns are the corresponding
    eigenvectors so that A*V = B*V*D.

    [V,D,W] = EIG(A,B) also produces a full matrix W whose columns are the
    corresponding left eigenvectors so that W'*A = D*W'*B.

    [...] = EIG(A,B,'chol') is the same as EIG(A,B) for symmetric A and
    symmetric positive definite B.  It computes the generalized eigenvalues
    of A and B using the Cholesky factorization of B.

    [...] = EIG(A,B,'qz') ignores the symmetry of A and B and uses the QZ
    algorithm. In general, the two algorithms return the same result,
    however using the QZ algorithm may be more stable for certain problems.
    The flag is ignored when A or B are not symmetric.

    [...] = EIG(...,'vector') returns eigenvalues in a column vector
    instead of a diagonal matrix.

    [...] = EIG(...,'matrix') returns eigenvalues in a diagonal matrix
    instead of a column vector.

    See also CONDEIG, EIGS, ORDEIG.

    Reference page in Doc Center
       doc eig

    Other functions named eig
```

```
codistributed/eig    DynamicSystem/eig    gpuArray/eig    sym/eig
```

Now you are ready to use the command with a matrix **A**

```
A = [2 -1  0
    -1  2 -1
    0  -1  2];
[V, L]=eig(A);
```

The 3x3 matrix **V** contains the three eigenvectors of **A** as its columns. The diagonal 3x3 matrix **L** contains the three eigenvalues of **A** on its diagonal.

Recall that eigenvalues and eigenvectors satisfy:

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

You can check these relations by computing $\mathbf{A}\mathbf{v} - \lambda\mathbf{v}$, which should be zero:

```
A*V(:,1)-L(1,1)*V(:,1)
```

```
ans = 3×1
10^-15 ×
   -0.1110
   -0.2776
   -0.1665
```

Here, V(:,1) is the first eigenvector of **A** (first column of **V**), L(1,1) is the first eigenvalue of A (first diagonal element of **L**). Note that we see again the familiar numerical error, i.e., $\mathbf{A}\mathbf{v} - \lambda\mathbf{v}$ is not exactly zero, but very nearly zero.

You can check that all three eigenvalues/eigenvectors satisfy this relation:

```
A*V(:,1)-L(1,1)*V(:,1)
```

```
ans = 3×1
10^-15 ×
   -0.1110
   -0.2776
   -0.1665
```

```
A*V(:,2)-L(2,2)*V(:,2)
```

```
ans = 3×1
10^-15 ×
         0
    0.1243
         0
```

```
A*V(:,3)-L(3,3)*V(:,3)
```

```
ans = 3×1
10^-15 ×
```

```
   -0.2220
         0
         0
```

You can also write all three relations in matrix form:

$$\mathbf{AV} = \mathbf{VL}$$

Check this numerically:

```
A*V-V*L
```

```
ans = 3×3
10^-15 ×
   -0.1110         0   -0.2220
   -0.2776    0.1243         0
   -0.1665         0         0
```

Diagonalization

To transpose a matrix, use the '. For example, to get the transpose of A, simply write:

```
A'
```

```
ans = 3×3
     2    -1     0
    -1     2    -1
     0    -1     2
```

If you multiply **V** and its transpose, you get the identity matrix:

```
V'*V
```

```
ans = 3×3
    1.0000   -0.0000   -0.0000
   -0.0000    1.0000    0.0000
   -0.0000    0.0000    1.0000
```

```
V*V'
```

```
ans = 3×3
    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

Note that, suprisingly, here $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T$. Thus, the inverse of **V** is its transpose, $\mathbf{V}^{-1} = \mathbf{V}^T$. Multiplying $\mathbf{AV} = \mathbf{VL}$ from the left by $\mathbf{V}^T$ thus gives

$$\mathbf{V}^T\mathbf{AV} = \mathbf{L}$$

Note that A is not diagonal, but after multiplying from the left and right by $\mathbf{V}^T$ and **V**, we obtain the diagonal matrix **L**. In this sense, the matrix **V**, diagonalizes the matrix **A**.

You can diagonalize in Matlab as follows:

```
V'*A*V
```

```
ans = 3×3
    0.5858   -0.0000   -0.0000
   -0.0000    2.0000    0.0000
   -0.0000    0.0000    3.4142
```

Normalization of eigenvectors

Recall from class that eigenvectors are only known up to an arbitrary multiplicative constant. That is, if **v** is an eigenvector, the $\alpha\mathbf{v}$ is also an eigenvector for any non-zero scalar α. You can thus create other eigenvectors of **A**:

```
b1 = 5*V(:,1);
b2 = -0.2*V(:,2);
b3 = 10*V(:,3);
B = [b1 b2 b3]; % the 3x3 matrix B has the three vectors b1, b2 and b3 as its columns
```

You can check that b1, b2 and b3 are indeed eigenvectors of **A**:

```
A*b1-L(1,1)*b1
```

```
ans = 3×1
10^-14 ×
   -0.0444
   -0.1332
   -0.1110
```

```
A*b2-L(2,2)*b2
```

```
ans = 3×1
10^-16 ×
         0
   -0.3597
         0
```

```
A*b3-L(3,3)*b3
```

```
ans = 3×1
10^-14 ×
         0
    0.3553
   -0.3553
```

Or in matrix form:

```
A*B-B*L
```

```
ans = 3×3
10^-14 ×
   -0.0444         0         0
   -0.1332   -0.0036    0.3553
   -0.1110         0   -0.3553
```

The inverse of **B**, however, is no longer its transpose:

```
B'*B
```

```
ans = 3×3
   25.0000    0.0000   -0.0000
    0.0000    0.0400   -0.0000
   -0.0000   -0.0000  100.0000
```

```
B*B'
```

```
ans = 3×3
   31.2700  -26.5165   31.2300
  -26.5165   62.5000  -26.5165
   31.2300  -26.5165   31.2700
```

But since $\mathbf{B}^T\mathbf{B}$ is still a diagonal matrix, **B** also diagonalizes **A**:

```
B'*A*B
```

```
ans = 3×3
   14.6447    0.0000   -0.0000
         0    0.0800   -0.0000
         0   -0.0000  341.4214
```

But note that $\mathbf{B}^T\mathbf{A}\mathbf{B}$ is no longer equal to **L**. It is equal to $\mathbf{B}^T\mathbf{B}\mathbf{L}$

```
B'*B*L
```

```
ans = 3×3
   14.6447    0.0000   -0.0000
    0.0000    0.0800   -0.0000
   -0.0000   -0.0000  341.4214
```

Since eigenvectors are only defined up to a multiplicative constant, it is common practice to "normalize" them so that $\mathbf{v}^T\mathbf{v} = 1$. This is what Matlab's eig command automatically does.

You can get the eigenvectors $\mathbf{v}$ that you computed using Matlab's back from the eigenvectors b1, b2, b3 by normalizing the eigenvectors such that $\mathbf{b}^T\mathbf{b} = 1$. You can achieve this by dividing $\mathbf{b}$ by $\sqrt{\mathbf{b}^T\mathbf{b}}$:

```
b1 = b1/sqrt(b1'*b1);
b1'*b1
```

```
ans = 1.0000
```

```
b2 = b2/sqrt(b2'*b2);
b2'*b2
```

```
ans = 1.0000
```

```
b3 = b3/sqrt(b3'*b3);
```

```
b3'*b3
```

```
ans = 1.0000
```

After this last step, the eigenvectors b1, b2, b3 are exactly the same as the eigenvectors contained in the matrix $\mathbf{V}$. There is, however, one more issue: the sign of and eigenvector is still not uniquely defined, even when one enforces that $\mathbf{v}^T\mathbf{v} = 1$. Define an eigenvector $\mathbf{u} = -\mathbf{v}$.

```
u=-V(:,1);
```

Then $\mathbf{u}^T\mathbf{u} = 1$ and $\mathbf{v}^T\mathbf{v} = 1$, but the vectors are clearly not the same.

```
u'*u
```

```
ans = 1.0000
```

```
V(:,1)'*V(:,1)
```

```
ans = 1.0000
```

```
u-V(:,1)
```

```
ans = 3×1
   -1.0000
   -1.4142
   -1.0000
```

Nonetheless, none of these non-uniqueness issues are critical to diagonalization. Do you know why?

Exercises

1. Why is $\mathbf{B}^T\mathbf{BL}$ (see above for definitions) a diagonal matrix?

Solution.

The matrix $\mathbf{B}^T\mathbf{B}$ is diaginal by its construction and so is the matrix **L**. Since the product of two diagonal matrices is also diagonal, $\mathbf{B}^T\mathbf{BL}$ is diagonal.

2 The determinant of a matrix **A** is equal to the product of the eigenvalues. Write a code that uses only the eig command and basic operations (such as multiplication and addition) to compute the determinant of the matrix

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

You can check your answer by using Matlab's det command. As usual, find out how to use the det command by using the help command.

Solution

Compute the eigenvalues of A.

```
[U,L]=eig(A);
```

The eigenvalues are the diagonal elements of L. We can thus compute det(A) by taking the product of the diagonal elements of A:

```
detA = L(1,1)*L(2,2)*L(3,3)
```

```
detA = 4.0000
```

We use Matlab's det command to check our result

```
det(A)
```

```
ans = 4
```

3. Compute the eigenvalues of the 1000x1000 matrix

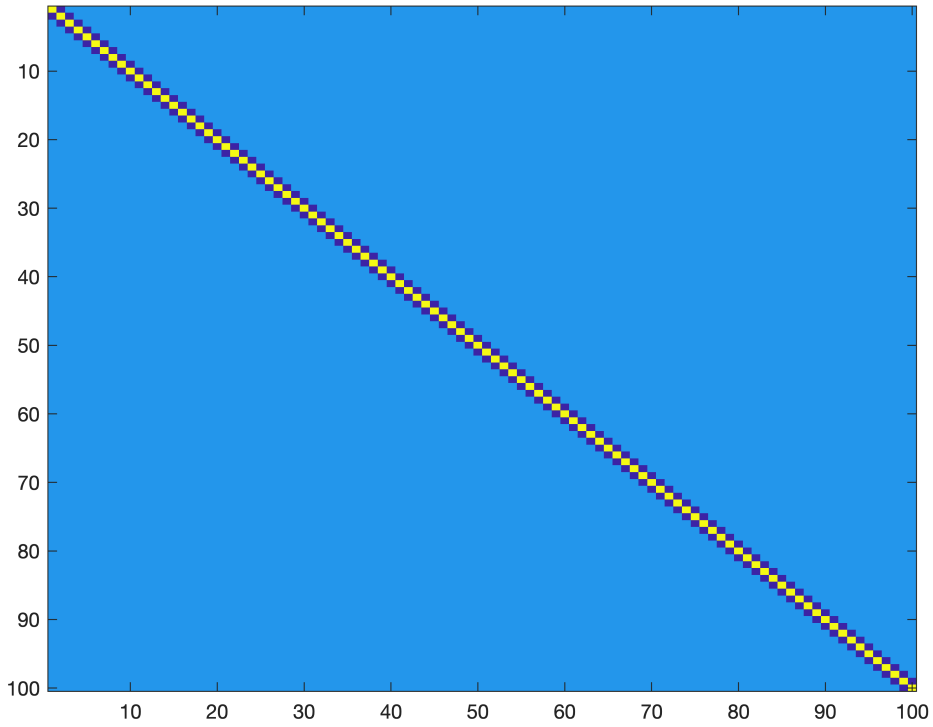
$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \cdots & & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \vdots \\ & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & 0 & -1 & 2 & -1 \\ & & & 0 & -1 & 2 \end{pmatrix}$$

You can use this code to create **A**

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

You can visualize such a large matrix by plotting the values of the elements as colors:

```
figure
imagesc(A)
```



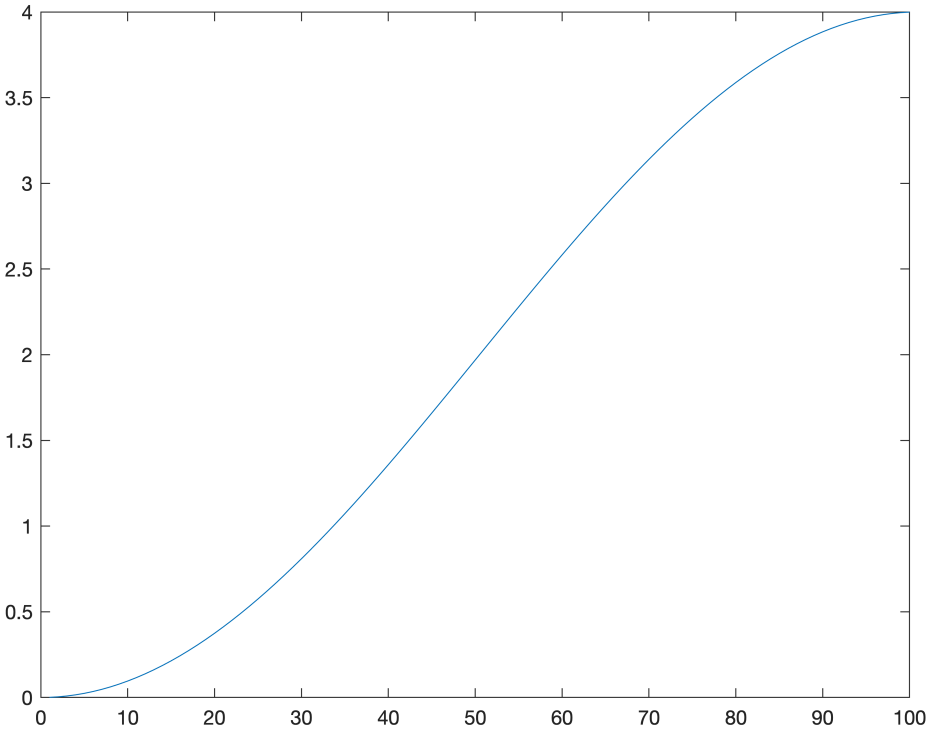
Solution

Use the eig command to compute the eigenvalues and eigenvectors of **A**.

```
[U,L]=eig(A);
```

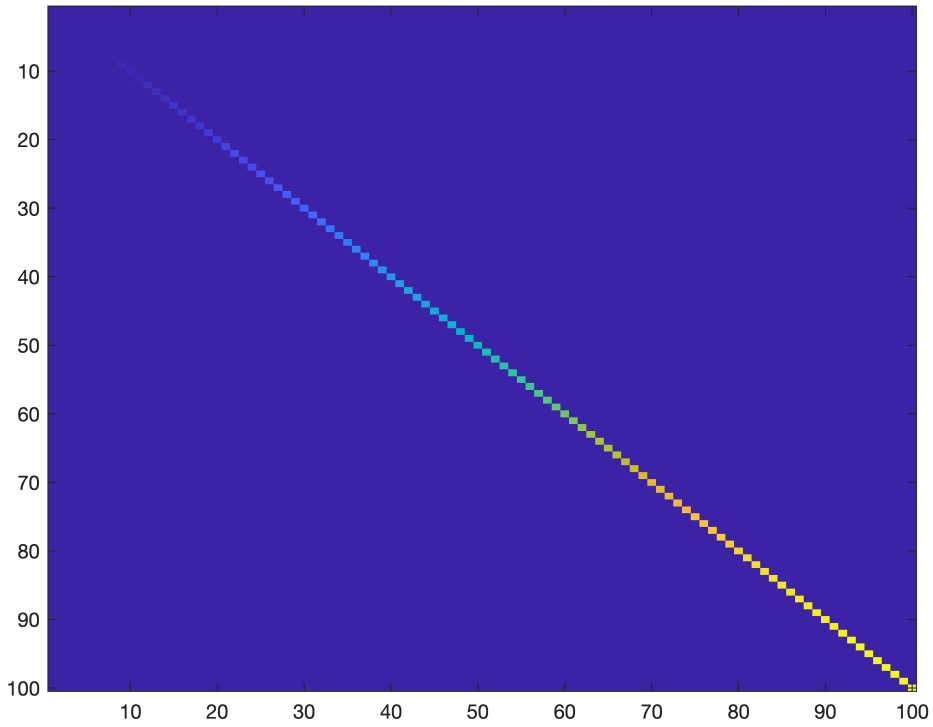
Make a plot of the eigenvalues by plotting the diagonal elements of the matrix **L** that you obtain fromt the eig command and that contains the eigenvalues of **A** on the diagonal. You can do this by using this command:

```
figure
plot(diag(L))
```



Diagonalize the matrix **A**. Make a plot of the diagonalized **A**.

```
Ad = U'*A*U;
figure
imagesc(Ad)
```


10
20
30
40
50
60
70
80
90
100
10
20
30
40
50
60
70
80
90
100