

## ssresp.eigcomp Computing eigendecompositions

1 Computing eigendecompositions is rather straightforward with a numerical or symbolic computing tool such as those available in Matlab or Python. The following sections show how to use Matlab and Python to compute numerical and symbolic eigendecompositions.

### Matlab eigendecompositions

Matlab numerical eigendecompositions

Consider the following matrix A.

```
A = [ ...
     -3, 5, 9; ...
     0, 2, -10; ...
     5, 0, -4 ...
     ];
```

What are its eigenvalues and eigenvectors? Let's use the MATLAB function eig. From the documentation:

$[V,D] = \text{EIG}(A)$  produces a diagonal matrix  $D$  of eigenvalues and a full matrix  $V$  whose columns are the corresponding eigenvectors so that  $AV = VD$ .

$$D = \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \end{bmatrix} \quad V = \begin{bmatrix} | & & \\ v_1 & & \\ | & & \end{bmatrix}$$

Let's try it.

```
[W,D] = eig(A);
disp(W);
disp(D);

[ -0.769, 0.122 + 0.537i, 0.122 + 0.537i]
[ 0.381, 0.767, 0.767]
[ 0.514, -0.0953 - 0.316i, -0.0953 + 0.316i]
```

The eigenvalues are on the diagonal of D.

```
disp(diag(D))

-11.487 + 0i
3.2433 + 4.122i
3.2433 - 4.122i
```

The eigenvectors are normalized to have unit length.

```
disp(norm(W(:,3))) % for instance

1
```

### Matlab symbolic eigendecompositions

Sometimes symbolic parameters in a matrix require symbolic eigendecomposition. In Matlab, this requires the `sympb` toolbox. First, declare symbolic variables.

```
syms a b c
```

Now form a symbolic matrix.

```
A = [ ...
     a,b; ...
     0,c; ...
     ]
```

```
A =
[ a, b]
[ 0, c]
```

The function eig is overloaded and if A is symbolic, the symbolic routine is called, which has a syntax similar to the numerical version above.

```
[W_sym,D_sym] = eig(A)

W_sym =
[ 1, -b/(a - c)]
[ 0, 1]
D_sym =
[ a, 0]
[ 0, c]
```

Again, the eigenvalues are on the diagonal of the eigenvalue matrix.

```
disp(diag(D_sym))

a
c
```

### Python eigendecompositions

Python numerical eigendecompositions

In Python, we first need to load the appropriate packages.

```
import numpy as np # for numerics
from numpy.linalg import la # for eig
from IPython.display import display, Markdown, Latex # pretty
np.set_printoptions(precision=3) # for pretty
```

Consider the same numerical A matrix from the section above. Create it as a numpy .array object.

```
A = np.array(
[
[-3, 5, 9],
[0, 2, -10],
[5, 0, -4],
])
```

The `numpy.linalg` module (loaded as `la`) gives us access to the `eig` function.

```
eigenvals,eigenvec = la.eig(A)
print("eigenvals: ",eigenvals)
print("eigenvec matrix: ",eigenvec)

eigenvals: [-11.487+0.j 3.243+4.122j 3.243-4.122j]
eigenvec matrix:
[[-0.769+0.j 0.122+0.537j 0.122+0.537j]
 [ 0.381+0.j 0.767+0.j 0.767+0.j ]
 [ 0.514+0.j -0.095-0.316j -0.095+0.316j]]
```

Note that the eigenvalues are returned as a one-dimensional array, not along the diagonal of a matrix as with Matlab.

```
print("the third eigenvalue is ",eigenvals[2].real)
```

the third eigenvalue is 3.243e+00-4.122e+00i  
Python symbolic eigendecompositions

We use the `sympy` package for symbolics.

```
import sympy as sp
```

Declare symbolic variables.

```
sp.var('a b c')
```

```
{a, b, c}
```

Define a symbolic matrix A.

```
A = sp.Matrix(
[ a,b],
[ 0,c]
)
display(A)
```

```
[ a b]
[ 0 c]
```

The `sympy.Matrix` class has methods `eigenvals` and `eigenvecs`. Let's consider them in turn.

```
A.eigenvals()
```

```
{a: 1, c: 1}
```

What is returned is a dictionary with our eigenvalues as its keys and the multiplicity (how many) of each eigenvalue as its corresponding value.

The `eigenvecs` method returns even more complexly structured results.

```
A.eigenvecs()
```

```
{a: 1, [Matrix([
[1],
[0]]]), c: 1, [Matrix([
[1/(a - c)],
[1]])]}
```

This is a list of tuples with structure as follows.

```
(eigenvalue, multiplicity, eigenvector)
```

Each eigenvector is given as a list of symbolic matrices.

Extracting the second eigenvector can be achieved as follows.

```
A.eigenvecs()[1][1][0]
```

```
[ -b
a-c
1]
```