Eigen Problems and Diagonalization Using Matlab

An Eigenproblem for a given $n \times n$ matrix $A$ requires finding the set of vectors, $x$, and the scalar numbers $\lambda$ such that

$$Ax = \lambda x.$$ 

In other words, we want the vectors which, when operated on by $A$, are simply multiples of the original vector. Geometrically, the eigenvectors of $A$ are those vectors, $x$, such that $Ax$ lies in the same (or exactly opposite) direction as $x$. $A$ simply multiplies its “own” (in German “eigen”) vectors. Multiplication by $A$ changes the direction of all other vectors.

Matlab allows for easy computation of the eigenvalues and eigenvectors of any square matrix. For example, consider the following Matlab commands:

```
> A = [-3 1 -3; -8 3 -6; 2 -1 2]
A =
     -3     1    -3
     -8     3    -6
     2    -1     2
```

To find the eigenvalues of $A$ we could use the fact that the eigenvalues, $\lambda$ satisfy the characteristic equation given by

$$\text{det}(A - \lambda I) = 0.$$ 

Matlab has an easy way of entering this. Simply use the `poly` command:

```
> p = poly(A)
p =
     1    -2    -1     2
```

The result says that the characteristic polynomial is:

$$p(\lambda) = \lambda^3 - 2\lambda^2 - \lambda + 2 = 0$$

This can be factored into:

$$(\lambda - 1)(\lambda + 1)(\lambda - 2)$$

Which gives us the eigenvalues of $A$ directly.

If you don’t see the factorization easily, Matlab is equipped to solve the characteristic equation for you using the `roots()` command,

```
> eigs = roots(p)
eigs =
     2
     1
     -1
```
which gives the zeros (eigenvalues) of the polynomial directly.

Now we can solve for the eigenvectors of \( A \). For each eigenvalue, we must solve

\[
(A - \lambda I)x = 0
\]

for the eigenvector \( x \). In Matlab the \( n \times n \) identity matrix is given by `eye(n)`. To find the eigenvector associated with \( \lambda = 2 \) we could use:

```matlab
> A1 = A - eigs(1)*eye(3) %Note: Use eigs(1) instead of '2' for accuracy
A1 =
    -5     1   -3
    -8     1   -6
     2    -1    0
```

```matlab
> rref(A1)
ans =
    1     0     1
    0     1     2
    0     0     0
```

This gives us \( x = \alpha \begin{pmatrix} -1 \\ -2 \\ 1 \end{pmatrix} \) The same procedure could be used for the other two eigenvectors.

Try it!

Seems complicated? Once again Matlab has a fast way of accomplishing the same task. The `eig()` command finds the eigenvalues and eigenvectors of a matrix directly. The output is given in two matrices. The first is a matrix whose columns contain the eigenvectors while the second is a diagonal matrix containing the eigenvalues.

```matlab
> [V,E] = eig(A)
V =
    881/2158    1292/2889   -780/1351
    881/1079    2584/2889   -780/1351
   -881/2158   *          780/1351

E =
    2     0     0
    0    -1     0
    0     0     1
```

If the output looks a bit strange, its because matlab normalizes the eigenvectors so that \((V_i \cdot \bar{V}_i) = 1\). For instance we can make the eigenvector corresponding to \( \lambda = 2 \) look like that given in our previous result:
> V1 = V(:,1)

V1 =
  881/2158
  881/1079
 -881/2158

> V1 = V1/V1(1)

V1 =
  1
  2
 -1

**Diagonalization:** Matlab’s eigenvector output format is exactly what we need to diagonalize the input matrix, namely a transformation matrix $P = V$ whose columns are the eigenvectors of $A$. To see the utility of diagonalization, consider the following set of nonhomogeneous, coupled ODEs

$$x' = Ax + F$$

where $x$ is the unknown vector of solutions and $A$ is matrix of constant coefficients.

To solve the coupled set of equations via diagonalization, we first transform to new variables, $y$ using the transformation matrix $V$:

$$x = V y$$

$$x' = V y' = Ax + F = AV y + F$$

In terms of the new variable, $y$,

$$y' = V^{-1} A V y + V^{-1} F$$

Since $V^{-1} A V$ is just the diagonal matrix of eigenvalues of $A$, this last set is completely uncoupled and easy to solve.

As an example, consider the coupled set of 1st order ODEs equivalent to the single 2nd order equation:

$$y'' + 3y' - 4y = 3e^{2t}$$

$$\begin{pmatrix} x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 4 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ 3e^{2t} \end{pmatrix}$$

Let’s solve the homogeneous 1st order problem using Matlab to do the matrix calculations.

First set up the matrix $A$ and find its transformation matrix.

> A = [0 1;4 -3]

A =
  0  1
  4 -3

> [v,d] = eig(A)

v =

$$\begin{pmatrix} \frac{881}{2158} \\ \frac{881}{1079} \\ -\frac{881}{2158} \end{pmatrix}$$
\[
\begin{array}{ll}
985/1393 & -528/2177 \\
985/1393 & 2112/2177
\end{array}
\]

d =
\[
\begin{pmatrix}
1 & 0 \\
0 & -4
\end{pmatrix}
\]

> v(:,1) = v(:,1)/v(1,1) \quad \text{%Note: Can multiply an eigenvector by a scalar}

\[
v =
\begin{pmatrix}
1 & -528/2177 \\
1 & 2112/2177
\end{pmatrix}
\]

Here we rescale the eigenvectors to make them 'prettier'

> v(:,2) = v(:,2)/v(1,2)

\[
v =
\begin{pmatrix}
1 & 1 \\
1 & -4
\end{pmatrix}
\]

We will also need the inverse, \( V^{-1} \):

> inv(v)

\[
\text{ans =}
\begin{pmatrix}
4/5 & 1/5 \\
1/5 & -1/5
\end{pmatrix}
\]

Now we have enough information to solve the problem. The uncoupled equations become:

\[
\begin{pmatrix}
y'_1 \\
y'_2
\end{pmatrix} =
\begin{pmatrix}
1 & 0 \\
0 & -4
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2
\end{pmatrix} +
\begin{pmatrix}
4/5 & 1/5 \\
1/5 & -1/5
\end{pmatrix}
\begin{pmatrix}
0 \\
3e^{2t}
\end{pmatrix}
\]

Or, individually,

\[
y'_1 - y_1 - 3e^{2t}/5 = 0
\]
\[
y'_2 + 4y_1 + 3e^{2t}/5 = 0
\]

The solution to these linear, 1st order ODEs are:

\[
(e^{-t}y_1)' = 3e^{t}/5
\]
\[
y_1 = c_1e^t + 3e^{2t}/5
\]

and

\[
(e^{4t}y_2)' = 3e^{6t}/5
\]
\[
y_2 = c_2e^{-4t} - 3e^{2t}/30
\]

To find the solution \( x \), simply transform back:

\[
x = V y =
\begin{pmatrix}
1 & 1 \\
1 & -4
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2
\end{pmatrix}
\]
\[
x = \begin{pmatrix}
    c_1 e^t + 3e^{2t}/5 + c_2 e^{-4t} - 3e^{2t}/30 \\
    c_1 e^t + 3e^{2t}/5 - 4c_2 e^{-4t} + 12e^{2t}/30
\end{pmatrix}
\]

\[
x = \begin{pmatrix}
    c_1 e^t + c_2 e^{-4t} + e^{2t}/2 \\
    c_1 e^t - 4c_2 e^{-4t} + e^{2t}
\end{pmatrix}
\]

**Matrix Powers by Diagonalization:** The work required to find the \( n^{th} \) power of a matrix is greatly reduced using diagonalization. As we showed in class,

\[
A^k = V D^k V^{-1}
\]

where \( V \) is the transformation matrix of \( A \) and \( D \) is the diagonal matrix of eigenvalues of \( A \). Therefore \( D^n \) is simply the diagonal matrix containing \( \lambda^k \) on the diagonal. For example, consider the following matrix:

\[
A = \begin{bmatrix}
    1 & 3 & 4 \\
    3 & -1 & 2 \\
    4 & 2 & 2
\end{bmatrix}
\]

The computationally fast way of calculating \( A^{10} \) is to use diagonalization.

\[
> [V,D] = eig(A)
\]

\[
V = \begin{bmatrix}
    0.7040 & -0.3182 & 0.6349 \\
    -0.6521 & -0.6437 & 0.4005 \\
    -0.2812 & 0.6959 & 0.6607
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
    -3.3764 & 0 & 0 \\
    0 & -1.6791 & 0 \\
    0 & 0 & 7.0555
\end{bmatrix}
\]

\[
> A10 = V * D^{10} * inv(V)
\]

\[
A10 = \begin{bmatrix}
    1.0e+008 * \\
    1.2330 & 0.7763 & 1.2819 \\
    0.7763 & 0.4911 & 0.8093 \\
    1.2819 & 0.8093 & 1.3347
\end{bmatrix}
\]

We can check by direct calculation:
\[ A^10 \]

\[
\begin{array}{ccc}
123304096 & 77633408 & 128193568 \\
77633408 & 49109984 & 80925664 \\
128193568 & 80925664 & 133474944 \\
\end{array}
\]

Which is exactly the same result. Note: Matlab probably performed the direct calculation using diagonalization anyway!