# MODIFIED VERSION OF: An introduction to **Matlab** for dynamic modeling

## ***PART 2 ***

Last compile: October 3, 2016

Stephen P. Ellner[1] and John Guckenheimer[2**]
[1]Department of Ecology and Evolutionary Biology, and
[2]Department of Mathematics
Cornell University

**** Modified by Eric Shea-Brown for AMATH 422/522:**
errors and other unpleasantries may have been introduced, unless you're in this class
refer to www.cam.cornell.edu/ $\sim$ dmb/DMBsupplements.html for the original!!
With additional material from Prof. Mark Goldman, UC Davis

# Introduction

These notes for computer labs accompany our textbook *Dynamic Models in Biology* (Princeton University Press 2006). They are based in part on course materials by former TAs Colleen Webb, Jonathan Rowell and Daniel Fink at Cornell, Professors Lou Gross (University of Tennessee) and Paul Fackler (NC State University), and on the book *Getting Started with Matlab* by Rudra Pratap (Oxford University Press). So far as we know, the exercises here and in the textbook can all be done using the Student Edition of Matlab, or a regular base license without additional Toolboxes.

| | |
|---|---|
| `zeros(n,m)` | $n \times m$ matrix of zeros |
| `ones(n,m)` | $n \times m$ matrix of ones |
| `rand(n,m)` | $n \times m$ matrix of Uniform(0,1) random numbers |
| `randn(n,m)` | $n \times m$ matrix of Normal($\mu = 0, \sigma = 1$) random numbers |
| `eye(n)` | $n \times n$ identity matrix |
| `diag(v)` | diagonal matrix with vector `v` as its diagonal |
| `linspace(a,b,n)` | vector of n evenly spaced points running from a to b |
| `length(v)` | length of vector v |
| `size(A)` | dimensions of matrix A [# rows, # columns] |
| `find(A)` | locate indices of nonzero entries in A |
| `min(A), max(A), sum(A)` | minimum, maximum, and sum of entries |

Table 1: Some important functions for creating and working with vectors and matrices; many more are listed in the Help system, Functions by Category:Mathematics:Arrays and Matrices. Many of these functions have additional optional arguments; use the Help system for full details.

# 1 Matrices

A matrix is a two-dimensional array of numbers. Matrices are entered as if they were a column vector whose entries are row vectors. For example:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

The values making up a row are entered with white space or commas between them. A semicolon indicates the end of one row and the start of the next one. The same process lets you combine vectors to make matrices. For example

```
>> L=1:3; W=2*L; B=[L;W;L]
```

creates `B` as a 3-row matrix whose $1^{st}$ and $3^{rd}$ rows are L=[1 2 3] and the $2^{nd}$ row is W=[2 4 6]. Similarly

```
>> C=[A, B]
```
or
```
>> C=[A B]
```
creates a matrix with 3 rows and 6 columns. As with vector creation, the comma between entries is optional.

MATLAB has many functions for creating and working with matrices (Table 1; Uniform(0,1) means that all values between 0 and 1 are equally likely; Normal(0,1) means the bell-shaped Normal (also called Gaussian) distribution with mean 0, standard deviation 1).

## Matrix addressing

Matrix addressing works like vector addressing except that you have to specify both row and column, or a range of rows and columns. For example `q=A(2,3)` sets q equal to 6, which is the ($2^{nd}$ row, $3^{rd}$ column) entry of the matrix **A**, and

```
>> v=A(2,2:3)
v =
     5     6
>> B=A(2:3,1:2)
B =
     4     5
     7     8
```

The Matlab Workspace shows that `v` is a row vector (i.e. the orientation of the values has been preserved) and `B` is a 2×2 matrix.

There is a useful shortcut to extract entire rows or columns, a colon with the limits omitted

```
>> firstrow=A(1,:)
firstrow =
     1     2     3
```

The colon is interpreted as "all of them", so for example `A(3,:)` extracts all entries in the $3^{rd}$ row, and `A(:,3)` extracts everything in the $3^{rd}$ column of A.

As with vectors, addressing works in reverse to assign values to matrix entries. For example,

```
>> A(1,1)=12
A =
    12     2     3
     4     5     6
     7     8     9
```

The same can be done with blocks, rows, or columns, for example

```
A(1,:)=rand(1,3)
A =
    0.9501    0.2311    0.6068
    4.0000    5.0000    6.0000
    7.0000    8.0000    9.0000
```

A numerical function applied to a matrix acts element-by-element:

```
>> A=[1 4; 9 16]; A, sqrt(A)
A =
     1     4
     9    16
ans =
     1     2
     3     4
```

The same is true for scalar multiplication and division. Try

```
>> 2*A, A/3
```

and see what you get.

If two matrices are the same size, then you can do element-by-element addition, subtraction, multiplication, division, and exponentiation:

```
A+B, A-B, A.*B, A./B, A.^B
```

**Exercise 1.1** Use `rand` to construct a 5×5 matrix of random numbers with a uniform distribution on $[0, 1]$, and then (a) Extract from it the second row, the second column, and the 3×3 matrix of the values that are not at the margins (i.e. not in the first or last row, or first or last column). (b) Use `linspace` to replace the values in the first row by `2 5 8 11 14`.

### Matrices and loading data

Another key step is often loading data from a text file. Get a copy of **ChlorellaGrowth.txt** and save it in your working directory to see how this is done. First, instead of having to type in the numbers at the Command line, the command

```
X=load('ChlorellaGrowth.txt')
```

reads the numbers in **ChlorellaGrowth.txt** and puts them into variable **X**. We extract them with the commands

```
Light=X(:,1); rmax=X(:,2);
```

As you know by now, these are shorthand for 'Light=everything in column 1 of X', and 'rmax=everything in column 2 of X'.

Type whos to confirm what you now have.

## 2 Matrix computations

### Matrix-vector multiplication

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x_1 \begin{pmatrix} A_{1,1} \\ A_{2,1} \end{pmatrix} + x_2 \begin{pmatrix} A_{1,2} \\ A_{2,2} \end{pmatrix}$$

e.g.

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

In general,

$$\begin{pmatrix} | & \cdots & | \\ a_1 & \cdots & a_n \\ | & \cdots & | \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \sum_j x_j \begin{pmatrix} | \\ a_j \\ | \end{pmatrix}$$

Also useful to think of equivalent "ROW-WISE" form of matrix multiplication.

MATLAB performs matrix-vector multiplication via the * operator.

```
>> A=[1 1 ; 1 2] ; A*[1 ; 2]

ans =

     3
     5
```

In $y = Ax$, $A$ must have same number of columns as $x$ has rows.

Nonsense:

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$$

```
>> A=[1 1 ; 1 2] ; A*[1 ; 2 ; 4]
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

**Exercise 2.1** : Compute the below by hand and check your result in MATLAB

$$\begin{pmatrix} 2 & -3 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

| | |
|---|---|
| `inv(A)` | inverse of matrix A |
| `det(A)` | determinant of matrix A |
| `trace(A)` | trace of matrix A |
| `poly(A)` | coefficients of characteristic polynomial |
| `expm(A)` | matrix exponential |
| `norm(A)` | Euclidean matrix norm |
| `find(A)` | locate indices and values of nonzero entries |
| `v=eig(A)` | vector of the eigenvalues of A, unsorted |
| `[W,D]=eig(A)` | diagonal matrix D of eigenvalues; matrix W whose columns are the corresponding eigenvectors |

Table 2: Some important functions for matrix computations. Many of these functions have additional optional arguments; use the Help system for full details.

One of Matlab's strengths is its suite of functions for matrix calculations. Some functions that we will eventually find useful are listed in Table 2; don't panic if you don't know what these are – they'll be defined when we use them.

Many of these functions only work on square matrices, and return an error if `A` is not square. **For the remainder of this section we only consider square matrices**, and focus on functions for finding their eigenvalues and eigenvectors.

We are often particularly interested in the dominant eigenvalue – the one with largest absolute value – and the corresponding eigenvector (the general definition of absolute value, which covers both real and complex numbers, is $|a+bi| = \sqrt{a^2+b^2}$). Extracting those from the complete set produced by **eig** takes some work. For the dominant eigenvalue:

```
>> A=[5 1 1; 1 -3 1; 0 1 3]; L=eig(A);
>> j=find(abs(L)==max(abs(L)))
>> L1=L(j);
>> ndom=length(L1);
```

In the second line `abs(L)==max(abs(L))` is a comparison between two vectors, which returns a vector of 0s and 1s. Then **find** extracts the list of indices where the 1's are.

The third line uses the "found" indices to extract the dominant eigenvalues. Finally, `length` tells us how many entries there are in `L1`. If ndom=1, there is a single dominant eigenvalue $\lambda$.

The **dominant eigenvector(s)** are also a bit of work.

```
>> [W,D]=eig(A)
>> L=diag(D)
>> j=find(abs(L)==max(abs(L)));
>> L1=L(j);
>> w=W(:,j);
```

The first line supplies the raw ingredients, and the second pulls the eigenvalues from D

into a vector. After that it's the same as before. The last line constructs a matrix with dominant eigenvectors as its columns. If there is a single dominant eigenvalue, then `L1` will be a single number and `w` will be a column vector.

To get the corresponding **left eigenvector(s)**, repeat the whole process on **`B=transpose(A)`**.

## Eigenvector scalings

The eigenvectors of a matrix population model have biologically meanings that are clearest when the vectors are suitably scaled. The dominant right eigenvector **w** is the stable stage distribution, and we are most interested in the relative proportions in each stage. To get those,

```
>> w=w/sum(w);
```

The dominant left eigenvector **v** is the reproductive value, and it is conventional to scale those relative to the reproductive value of a newborn. If newborns are class 1:

```
>> v=v/v(1);
```

**Exercise 2.2**: Write an m-file which applies the above to A=[1 5 0; 6 4 0; 0 1 2]. Your file should first find **all** the eigenvalues of A, then extract the dominant one and the corresponding (right) eigenvector, scaled as above. Repeat this for the transpose of A to find the dominant left eigenvector, scaled as above.

## 2.1 Eigenvalue sensitivities and elasticities

**DON'T WORRY ABOUT THE MATERIAL BELOW UNTIL WE COVER SENSITIVITIES IN CLASS ...**

For an $n \times n$ matrix `A` with entries $a_{ij}$, the sensitivities $s_{ij}$ and elasticities $e_{ij}$ can be computed as

$$s_{ij} = \frac{\partial \lambda}{\partial a_{ij}} = \frac{v_i w_j}{\langle v, w \rangle} \qquad e_{ij} = \frac{a_{ij}}{\lambda} s_{ij} \tag{1}$$

where $\lambda$ is the dominant eigenvalue, **v** and **w** are dominant left and right eigenvalues, and $\langle v, w \rangle$ is the inner product of **v** and **w**, computed in Matlab as `dot(v,w)`. So once $\lambda, v, w$ have been found and stored as variables, it just takes some for-loops to compute the sensitivities and elasticities.

```
n=length(v);
vdotw=dot(v,w);
for i=1:n; for j=1:n;
    s(i,j)=v(i)*w(j)/vdotw;
end; end;
e=(s.*A)/lambda;
```

Note how the elasticities are computed all at once in the last line. In Matlab that kind of "vectorized" calculation is **much** quicker than computing them one-by-one in a loop. Even faster is turning the loops into a matrix multiplication:

```
vdotw=dot(v,w);
s=v*w'/vdotw;
e=(s.*A)/lambda;
```

**Exercise 2.3** Construct the projection matrix **A**, and then find $\lambda, v, w$ for an age-structured model with the following survival and fecundity parameters:

- Age-classes 1-6 are genuine age classes with survival probabilities

  $$(p_1, p_2, \cdots, p_6) = (0.3, 0.4, 0.5, 0.6, 0.6, 0.7)$$

  Note that $p_j = a_{j+1,j}$, the chance of surviving from age $j$ to age $j+1$, for these ages. You can create a vector `p` with the values above and then use a for-loop to put those values into the right places in `A`.

- Age-class 7 are adults, with survival 0.9 and fecundity 12.

**Results**: $\lambda = 1.0419$

$w = (0.6303, 0.1815, 0.0697, 0.0334, 0.0193, 0.0111, 0.0547)$
$v = (1, 3.4729, 9.0457, 18.8487, 32.7295, 56.8328, 84.5886)$

IF YOUR RESULTS TO NOT MATCH, READ ON ...

According to the lab manual, you should get for the dominant eigenvalue, and dominant left and right eigenvectors $v$ and $w$ above.

However, you will probably find that MATLAB returns:

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

V =

   0.8711            0.8711            0.8852            0.8852            0.9158            0.9158            0.9506
  -0.3301 - 0.1753i  -0.3301 + 0.1753i  -0.0387 - 0.3626i  -0.0387 + 0.3626i   0.2552 - 0.2174i   0.2552 + 0.2174i   0.2737
   0.1198 + 0.1771i   0.1198 - 0.1771i  -0.1958 + 0.0423i  -0.1958 - 0.0423i   0.0260 - 0.1615i   0.0260 + 0.1615i   0.1051
  -0.0162 - 0.1520i  -0.0162 + 0.1520i   0.0431 + 0.1306i   0.0431 - 0.1306i  -0.0518 - 0.0853i  -0.0518 + 0.0853i   0.0504
  -0.0489 + 0.1217i  -0.0489 - 0.1217i   0.1032 - 0.0467i   0.1032 + 0.0467i  -0.0694 - 0.0230i  -0.0694 + 0.0230i   0.0290
   0.0860 - 0.0726i   0.0860 + 0.0726i  -0.0473 - 0.0805i  -0.0473 + 0.0805i  -0.0496 + 0.0201i  -0.0496 - 0.0201i   0.0167
  -0.0448 + 0.0238i  -0.0448 - 0.0238i  -0.0057 + 0.0534i  -0.0057 - 0.0534i   0.0476 + 0.0406i   0.0476 - 0.0406i   0.0825


D =

  -0.6176 + 0.3279i       0                 0                 0                 0                 0                 0
        0            -0.6176 - 0.3279i       0                 0                 0                 0                 0
        0                 0            -0.0773 + 0.7241i       0                 0                 0                 0
        0                 0                 0            -0.0773 - 0.7241i       0                 0                 0
        0                 0                 0                 0            0.6239 + 0.5314i       0                 0
        0                 0                 0                 0                 0            0.6239 - 0.5314i       0
        0                 0                 0                 0                 0                 0            1.0419
```

Here, the diagonal entries of $D$ are the eigenvalues. The corresponding columns of $D$ are the associated eigenvectors. In particular the dominant eigenvalue is the last one in the

list, $\lambda = 1.0419$. That checks against the Ellner and Guckenheimer lab manual. However, according to MATLAB the corresponding right eigenvector is the last column of $V$, which is

```
>> w=V(:,7)

w =

    0.9506
    0.2737
    0.1051
    0.0504
    0.0290
    0.0167
    0.0825
```

That does NOT match the given answer. What is going on ?

**We need to remember that eigenvectors are defined only up to an arbitrary scale factor.**

The defining equation for $x$ to be an eigenvector of $A$ with eigenvalue $\lambda$ is

$$Ax = \lambda x \ .$$

But note that if $x$ is an eigenvector with eigenvalue $\lambda$, so is $c\,x$, where $c$ is any constant (even a negative number, say). Indeed, it passes the same test:

$$\begin{aligned} A(cx) &= c\,Ax \\ &= c\,\lambda\,x \\ &= \lambda\,(cx) \end{aligned}$$

If we use $c = 0.6303/0.9506$, we see that our eigenvector agrees with the one in the lab manual. That is,

```
>> w*0.6303/0.9506

ans =

    0.6303
    0.1815
    0.0697
    0.0334
    0.0193
    0.0111
    0.0547
```

which DOES give us the answer in the lab manual.

The point is that even though we talk about “the” dominant eigenvector, etc., this is only meaningful up to a scale constant. But, it is still useful:

1. The ratios of the entries in the dominant eigenvector are unaffected by scaling, and represent relative proportions in the various population stages.
2. In our formula for eigenvalue sensitivities, the scale constant will cancel out in the numerator and denominator – so we always get the same answer.