# Octave tutorial

by Ross Coleman

Octave is open-source software mostly compatible with proprietary Matlab. It is not a computer algebra system, but it is especially good at doing linear algebra with numbers (not true variables) and doing all sorts of numerical computations.

**Other great sources of documentation on Octave:**

Official Octave manual
http://www.gnu.org/software/octave/doc/interpreter/

Dr. Bindner's tutorial on computer algebra systems (including Octave ironically)
http://limestone.truman.edu/ClassWiki/RosettaStone/

## Using variables & basic commands

Note: In the style of an Octave script, I'm adding comments preceded by a pound character.

```
x=1             #Output: x=1
y=1;            #No output
printf("The semicolon hides output!\n");     #Output: The semicolon hides output!
x               #Output: x = 1
y               #Output: y = 1
```

I will quit adding comments on the expected output.
Notice that we can change x and y from numeric types to strings without problems.

```
x="A string"
y='Octave variables do not have fixed types!'
```

The built in variable “ans” holds the most recent output that was not explicitly assigned to a variable.

```
3*4+1
ans     #This was the output from the line above.
x=23
ans     #This is still the output of 3*4+1.
4^3
ans     #Now “ans” has changed.
```

## Vectors and Matrices

```
x = [1,2,3]     #A row vector;
y = [4;5;6]     #A column vector;
x*y             #A dot product (one method)
A = [1,2;3,4]   #A matrix
A'              #Transpose of the matrix
B = [5,6;7,8]
A*B             #Matrix multiplication
A.*B            #Element by element mulitplication
A(2,1)          #element in row 2, column 1 of matrix A
printf("That was the element in row 2, column 1 of matrix A.\n");
x(1,2)          #element in row 1, column 2 of row vector x
x(2)            #second element of row vector x -- same!
y(3,1)          #element in row 3, column 1 of column vector y
```

```
y(3)            #third element of column vector y -- same!
```

Note: Octave is very powerful with linear algebra. We will skip most of that, however.

**Plotting functions**

```
x=1:10          #x equals a row vector containing the integers from one to ten
```

Note: x^2 can't be computed because x*x is undefined (can't do the matrix multiplication)

```
x.^2            #square of elements of x -- This is what we want!
y=x.^2
plot(x,y)       #plot of y=x^2 for x = 1,2,3,...,10
```

Or we can can skip a step.

```
plot(x,x.^2)    #same plot
```

That looks pretty good.

```
plot(x,x.^6)
```

That doesn't. Often we need more points than just the integers.

```
y=linspace(1,10,25)   #y equals a row vector of 25 elements starting at 1, going to 10
```

Notice how that fills up the screen? Often, we'll want to use 10,000 element row vectors or even bigger. Usually we will use “linspace” with a semicolon to suppress output, but here we want to see how it works.

```
y=linspace(10,1,25)   #This one counts down!
plot(y,y.^6)          #This plot is better!
plot(x,x.^6)          #Notice that the plot is replaced even if you don't close the window.
hold on               #This command will allow us to add multiple plots to the same window
plot(y,y.^6)
plot(y,exp(y))        #We're still adding plots to the same window
hold off
plot(y,sin(y))        #Now we're not.
plot(x,sin(x))        #That really looks bad.
t=linspace(0,4*pi,10000);
```

If you try the previous expression without the semicolon, use q to quit "less" (the program that lets you scroll through output).

```
plot(t,sin(t))        #This one is beautiful!
```

**Creating your own functions**

Helpful reference: http://linuxgazette.net/112/odonovan.html

Say we want a function that prints “This is a useless function.” to the screen.

```
function useless
        printf(“This is a useless function\n”);
endfunction
```

Try the function out. The '\n' gives a newline character (enter).

```
useless
```

Now say we want a function named “twopi” that returns the value of two times pi.

```
function retval =  twopi
        retval = 2*pi;
endfunction
```

This function sets “retval” (what we named our return value) equal to the value of 2 times pi. In other words, it returns the value of two times pi. Try it.

```
twopi
```

We could name our return value anything.

```
function stupidvariablename = twopi
```

```
        stupidvariablename = 2*pi;
    endfunction
    twopi
```

Now suppose we want to give a function input and return some kind of output. For example, our function "doubler" will double any numerical value (or matrix), it is given.

```
    function retval = doubler(x)
        retval = 2*x;
    endfunction
```

Let's try it out on a scalar and a matrix.

```
    doubler(3.14)
    doubler([1,0;5,2.3])
```

In general, we could have a function of the form

```
    function [retval1, retval2, etc.] = name(arg1, arg2, etc.)
```

where we can have mulitple parameters and multiple return values for the function. Note that variable types are flexible in Octave. Suppose we want "poly3xy" to take two variables, x and y, and return the row vector

[x^3 x^2*y x*y^2 y^3]

We define the function as follows

```
    function retval = poly3xy(x,y)
        retval(1)=x^3;
        retval(2)=x^2*y;
        retval(3)=x*y^2;
        retval(4)=y^3;
    endfunction
```

Trying "retval(2,3)" we get [8 12 18 27].

## Solving Differential Equations

Our systems of differential equations will be in the form:

$$\begin{aligned}\frac{dx_1}{dt} &= f_1(x_1, x_2, .., x_n, t)\\ \frac{dx_2}{dt} &= f_2(x_1, x_2, .., x_n, t)\\ &\vdots\\ \frac{dx_n}{dt} &= f_n(x_1, x_2, .., x_n, t)\end{aligned}$$

The equations give a first order system for which each equation can be solve for the derivatives. We will solve the system numerically with "lsode." (Octave also has a function "daspk" to solve differential algebraic equations, for more general systems that can't be put in this form.) Since $d/dt$ is a linear operator, we can rewrite this system as

$$\dot{X} = \frac{dX}{dt} = \frac{d}{dt}\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} f_1(x_1, x_2, .., x_n, t) \\ f_2(x_1, x_2, .., x_n, t) \\ \vdots \\ f_n(x_1, x_2, .., x_n, t) \end{bmatrix} = f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, t\right) = f(X, t)$$

where we have defined a new function $f$ that takes a vector of dependent variables and a scalar independent variable to replace the vector of scalar functions. The function "lsode" needs $f$, an initial value for $X$ which we will call $X_0$, and a row vector $t$, in which the first element corresponds to $X_0$. Suppose we have the nonlinear system

$$\frac{dx}{dt}=\frac{x}{y} \quad x(0)=1$$
$$\frac{dy}{dt}=xy \quad y(0)=1$$

Applying $\frac{dy}{dx}=\frac{dy/dt}{dx/dt}$ , we can show the solution is given by $y=1/(2-x)$ , $x=e^t\,sech\,t$ , $y=e^t\cosh t$ . Let's try it in Octave.

```
function xdot = f(x,t)
        xdot(1)=x(1)/x(2);
        xdot(2)=x(1)*x(2);
        #Note the use of subscripts to specify elements in vectors xdot and x
endfunction
t=linspace(0,3,10000);
A=lsode("f",[1;1],t);
x=A*[1;0];
y=A*[0;1];
```

Note: A has the form

$$A=\begin{bmatrix} x(t_1) & y(t_1) \\ x(t_2) & y(t_2) \\ \vdots & \vdots \end{bmatrix}$$

so we have multiplied by the vectors $\vec{e}_1=\begin{bmatrix}1\\0\end{bmatrix}$ and $\vec{e}_2=\begin{bmatrix}0\\1\end{bmatrix}$ to get the columns separately.

Now let's try some plots to see if we have found the correct answers.

```
plot(t,x)
hold on
plot(t,exp(t).*sech(t))
hold off
plot(t,y)
hold on
plot(t,exp(t).*cosh(t))
hold off
plot(x,y)
hold on
plot(x,1./(2-x))
```

On all three plots, the second line should overlap the first line, showing that "lsode" has indeed found the correct solution. This is a simple case that we can solve exactly, so it is useful to check how good Octave is at solving equations and whether you are using Octave correctly.

Now consider the following system

$$\frac{dx}{dt}=Ax+By$$
$$\frac{dy}{dt}=Cxy$$

It would be nice to allow a user to pass values for A, B, and C into the definition of the function *f*. However, "lsode" can only use functions that take two parameters, a vector and a scalar, and returns a vector. That means that we can't use "function xdot = f(x,t,A,B,C)." Even passing references or pointers into the function doesn't work. (I'm pretty sure.) We have to either initialize the parameters A,B, and C within the definition of *f* or use global variables. What if we want to be able to check the

behavior over a range of values for B? We can put the "lsode" execution in a loop in which we increment B. If we initialize B inside the function definition, however, incrementing B in the loop will have no effect. Thus, we have to use global variables, even though using them is usually considered a poor programming practice.

```
function retval = myquadratic(x)
        A = 1.5;
        B = 2;
        retval = A*x^2+B*x;
endfunction
myquadratic(2)
A=3;
myquadratic(2)
```

Notice that the value of "myquadratic(2)" doesn't change! The local variable A in the function is different from the variable A in the scope of the main program. That means that changing the value of A outside the function has no effect on the A outside the function, and vice versa.

```
clear all
```

**Note: "clear all" deletes all functions and variables from memory. "clear A" deletes just variable A.**

Interestingly, Octave requires global variables to be declared within the function definition.

```
global A = 1.5;
global B = 2;
function retval = myquadratic(x)
        global A;       #These 2 lines must be included for the function to use global variables.
        global B;
        retval = A*x^2+B*x;
endfunction
myquadratic(2)
A=3;
myquadratic(2)
```

Notice that the value of "myquadratic(2)" changes. This is because global variables can be accessed and changed by the main program or by its functions.

This is particularly useful when we want to see behavior over a range of parameters.

```
clear all
global A;
global B = 2;
function retval = myquadratic(x)
        global A;
        global B;
        retval = A*x^2+B*x;
endfunction
for A = 1:10
        myquadratic(2)
endfor
```

**Note: In most cases you would want to use**

```
function retval = myquadratic(x,A,B)
        retval=A*x^2+B*x;
endfunction
for A = 1:10
```

```
        myquadratic(2,A,2)
    endfor
```

**however, "lsode" can not handle such a function, so global variables are the best option if the function is the right hand side of a first order differential equation.**

## Octave Scripts: a simple SIR model

Consider a simple SIR model of an epidemic.

$$\frac{dS}{dt} = -rSI \qquad S(0) = S_0$$

$$\frac{dI}{dt} = rSI - \gamma I \qquad I(0) = I_0$$

$$\frac{dR}{dt} = \gamma I \qquad R(0) = 0$$

Let $\rho = \gamma / r$ . It can easily be shown that $I = N + \rho \ln(S/S_0) - S$ where N = S + I + R. The following script allows a user to enter values for $S_0, I_0, \gamma, r, t_0, t_f$ and the number of steps to use in the computation and get S(t), I(t), R(t) and a plot of I vs S.

```
S0 = input("S0: ");
I0 = input("I0: ");
global r;
r = input("r: ");
global gamma;
gamma = input("gamma: ");
tLow = input("Minimum time: ");
tHigh = input("Maximum time: ");
numsteps = input("Number of steps: ");
t=linspace(tLow,tHigh,numsteps);
function xdot = f(x,t)
        global r;
        global gamma;
        xdot(1) = -r*x(1)*x(2);
        xdot(2) = r*x(1)*x(2)-gamma*x(2);
endfunction
x0=[S0;I0];
A=lsode("f",x0,t);
S=A*[1;0];
I=A*[0;1];
R=(S0+I0)-S-I; #because N=S0+I0
[t',S,I,R]        #this may fill up several screens... scroll with arrows or type q to escape
plot(S,I)
hold on
#plot(S,S0+I0-S+(gamma/r)*log(S/S0))
#above line affected by "numpoints" choice -- produces a poor graph if "numpoints" is small
x=linspace(min(S),max(S),1000);
plot(x,S0+I0-x+(gamma/r)*log(x/S0)) #graphs should coincide if numsteps is high enough
printf("Global variables r and gamma are now being cleared!\n");
clear r;
clear gamma;
hold off
```

Use a text editor to create a file named “SIR.m.” (The *.m extension is required for Octave scripts.) Enter the code shown above and save the file. Within Octave type “SIR” (without the *.m). You will be prompted for several values. Use S0 = 990, I0 = 10, r = .5, gamma = 50, tLow = 0, tHigh = 2, and numsteps = 1000. Try changing your choice of “numpoints” and see how the result compares with the exact solution. Clearing global variables is important if you want the script to prompt for them again if you run the script a second time within Octave. The use of min(S) and max(S) is just what it sounds like here. The functions “min” and “max” have more power than what we just used, however. We used “[t',S,I,R]” with the transpose of “t” because “t” is a row vector whereas “S,” “I,” and “R” are column vectors.

It is also possible to put function definitions in a file. Note that Octave cannot handle nested function definitions. In other words, we can't write the following:

```
function redgreen
        function green
                printf(“green\n”)
        endfunction
        printf(“red”)
        green
endfunction
```

However, we can write the following

```
function redgreen
        printf(“red”)
        green
endfunction
function green
        printf(“green\n”)
endfunction

redgreen        #prints “redgreen\n” to the screen
```

We can put the following code in a file named “*functionname*.m” and must save somewhere in Octave's search path. The function specified by the filename must be defined first in the file. Any functions it calls must be defined below it. Thus, we can create “SIRfunction.m” and enter the following code

```
function retval = SIRfunction(S0,I0,rparam,gammaparam,tLow,tHigh,numsteps)
        global r=rparam;
        global gamma=gammaparam;
        t=linspace(tLow,tHigh,numsteps);
        x0=[S0;I0];
        A=lsode("f",x0,t);
        S=A*[1;0];
        I=A*[0;1];
        R=(S0+I0)-S-I; #because N=S0+I0
        retval = [t',S,I,R];
endfunction

function xdot = f(x,t)
        global r;
        global gamma;
        xdot(1) = -r*x(1)*x(2);
        xdot(2) = r*x(1)*x(2)-gamma*x(2);
```

```
endfunction
```

Now if we execute the following command, we will get [t',S,I,R] for the specified parameters.

```
SIRfunction(990,10,.5,50,0,2,1000)
```

The great thing about this is that our "global" variables we define in "SIRfunction.m" are "global" to the file not to all of Octave. The downside is that there is a performance hit when we call functions defined in files, according to the official Octave documentation. Octave checks the time-stamp of the files that define functions every time those functions are called to be sure that the functions haven't changed. Checking the time-stamp every time a function is called, slows down program execution. The other downside is that we have to remember the order of the parameters when we want to call the function. On the whole, I think it is best to put everything in one Octave script file and execute the script, so long as the script file doesn't grow so long as to be unmanageable.

When we prompted the user for values, we were making the program user-friendly. One didn't have to understand SIR.m code to use it. However, we make the code most programmer friendly if we put the parameter values right into the Octave script. Then we don't have to input parameters by hand every time we want to execute the script to check our work.

**Looping over parameters to get Imax and the corresponding t.**

Enter the following row vector into Octave.

```
x=[1,25,-6,pi,100, 2.31]
```

Now let's find the largest element in x and its index.

```
[xmax,i] = max(x)
x(i)      ## should equal xmax
```

This will come in useful when we want to find the largest value of I, and the corresponding values for t, S, V, and R.

```
[Imax,imax]=max(I);
tmax=t(imax);
Smax=t(imax);
...
```

If we loop over a parameter, it would be nice to save the parameter, tmax, and Imax for each value of the parameter. We could construct a three column table of those values to see how changing a parameter affects the intensity of an epidemic and the time to reach the maximum number of infectives, both of which are useful for vaccine policy. We could save values by appending another row onto a linked list for every value of the parameter. However, if we are looping with a definite number of iterations, it is more efficient to save to a matrix or vector of predetermined size. We will want to use subscripts to access individual elements of three vectors for the parameter, tmax, and Imax. We want to be able to create "empty" vectors of the given size before we actually set the entries, one by one. The function "zeros(rows,columns)" will create matrices with a zero in each entry.

```
zeros(3,5)       #produces a 3x5 matrix with zeros for entries
```

Octave's while and for loops work as shown below.

```
n=25
i=1
while i<=sqrt(n)
        disp("A while loop.")          #an alternative to C-style "printf(x)"
        i++;
endwhile
##notice that "disp()" adds newline characters automatically
for i=1:5
        disp("A for loop.")
endfor
```

Consider the model given in "A Vaccination Model for Transmission Dynamics of Influenza." After performing a normalization, we can remove the dR/dt equation and get the following system.

$$\frac{dS}{dt} = (1-\epsilon) - \beta SI - \xi S - S + \omega V + \delta(1 - S - V - I)$$
$$\frac{dV}{dt} = \xi S - (1-\sigma)\beta VI - (1+\omega)V$$
$$\frac{dI}{dt} = \epsilon + \beta SI + (1-\sigma)\beta VI - (1+\alpha)I$$

Here is Octave code to solve the system with parameter values indicated in the code, looping over parameter beta.

```
S0 = 8/7;
V0 = 1.8/7;
I0 = 1/35;
x0 = [S0;V0;I0];

global alpha = 1820;
global beta = 3430;    #The value we initialize here won't matter.  This was my best guess at a
                       #reasonable value for beta.
global delta = 70;
global epsilon = .3;
global xi = 35;
global sigma = .6;
global omega = 70;

tLow = 0;
tHigh= 10;
numsteps = 100000;

t=linspace(tLow,tHigh,numsteps);

function xdot = f(x,t)
        global alpha;
        global beta;
        global delta;
        global epsilon;
        global xi;
        global sigma;
        global omega;
        xdot(1)=(1-epsilon)-beta*x(1)*x(3)-xi*x(1)-x(1)+omega*x(2)+delta*(1-x(1)-x(2)-x(3));
        xdot(2)=xi*x(1)-(1-sigma)*beta*x(2)*x(3)-(1+omega)*x(2);
        xdot(3)=epsilon+beta*x(1)*x(3)+(1-sigma)*beta*x(2)*x(3)-(1+alpha)*x(3);
endfunction

beta = 0;
deltabeta = 100;
betasteps=200;
Beta=zeros(betasteps,1);
Imax=zeros(betasteps,1);
```

```
tmax=zeros(betasteps,1);

printf("We are solving the system of ODE's for 200 values of beta.\n******This may take
several minutes******\n");
fflush(stdout); #flushes output to the screen before continuing on with the script

for ibeta = 1:betasteps
        X = lsode("f",x0,t);
        S=X*[1;0;0];
        V=X*[0;1;0];
        I=X*[0;0;1];
        [Imax(ibeta),im]=max(I);
        Beta(ibeta)=beta;
        tmax(ibeta)=t(im);
        beta = ibeta*deltabeta;
endfor

printf("You may wish to type \"[Beta,tmax,Imax]\" Type q to stop output.\n");
printf("\"plot(Beta,tmax)\" and \"plot(Beta,Imax)\" are interesting.  You may wish to try them as
well.\n");
```

**Saving and Loading Files**

The script we just ran above probably took several minutes to run. It would be nice to save it's output so that we don't have to run it every time we want to see the system's beta dependence again. Octave allows for C style input and output to the screen and to files. It also has functions to input and output more convenient native Octave files.

save [options] file [v1 v2 ...] is the syntax for saving Octave files. Useful options include

```
-binary         #Octave's native binary format
-text           #Octave's native text format
-z/-zip         #gzip the file
```

[v1 v2 ...] is an optional list of variables. If none are specified, all variables are saved to the file. Note that wildcard characters are often useful.

load [options] file [v1 v2 ...]
Options are the same for load. If no variables are specified, all variables in the file will be loaded.

If you still have output from the script we ran above that loops over values of Beta, try saving the following files.

```
save fluOutput Beta tmax Imax
BtI = [Beta, tmax, Imax];
save fluMatrix BtI
```

To find out where you just saved the files type

```
pwd     #print working directory
```

Look at the files in a text editor. Notice that “fluMatrix” is very close to a tab seperated values file that can be read by a spreadsheet program. Using gedit I can turn it into a TSV file by first deleting the comment lines, doing a find and replace to replace all “\n ” with “\n”, deleting the first space character in the document, and replacing all “ ” (space characters) with “\t” (tab character). Or we can create CSV (comma seperated value) files by replacing all “ ” with “,”. Most spreadsheet programs will open either one. I created a CSV file named fluMatrix.csv. In Octave, I issued the command

```
clear all       #delete all variables so that we can be sure we are looking at what we imported
```

```
A = load "fluMatrix.csv";
A
```

Notice that Octave imports a CSV file as a matrix. On the other hand we can import the native Octave files as follows.

```
import fluOutput
import fluMatrix
```

Compare matrix "BtI" with "A."

```
BtI
A
BtI(24,3)
BtI(24,3)-.11175
A(24,3)
A-.11175
```

Notice that both have many significant figures. For a while I thought that importing a CSV could cause some significant figures to be lost, but I think I was mistaken. I re-tried importing the same matrix from an Octave file and from a CSV file, and I get the same results. This means that one can fairly easily import and export data to a spreadsheet program.

Saving Octave plots: from Dr. Bindner's documentation
from http://limestone.truman.edu/~dbindner/calc3/plotting.pdf

```
gset terminal postscript eps
gset output "filename.ps"
replot
```

That will save a plot as an encapsulated postscript file. Eps files are nice for putting in other postscript documents, LATEX documents, and pdf documents. Eps is a vector graphics format, so enlarging a plot is not a problem.

## Epidemic Models with Age Classes

Models with age classes will have $\vec{S}$ instead of $S$, $\vec{I}$ instead of $I$, and $\vec{R}$ instead of $R$. The third entry in $\vec{S}$ gives the number of susceptibles in the third age class. Our differential equations will require that the function we define to send to "lsode" will take the vector

$$\begin{bmatrix} \vec{S} \\ \vec{I} \end{bmatrix} \text{ and return } \begin{bmatrix} d\vec{S}/dt \\ d\vec{I}/dt \end{bmatrix}$$

from the function. A simplified example of such a function is shown below. Say we have three age classes.

```
function xdot = func(x,t)
    S=zeros(3,1);
    I=zeros(3,1);
    for i=1:3
        S(i)=x(i)
        I(i)=x(i+3)
    endfor
    for i=1:3
        xdot(i)=S(i)^i;
        xdot(i+3)=I(i)^-i;
    endfor
endfunction
func([2;2;2;2;2;2;)     #output: [2^1,2^2,2^3,2^-1,2^-2,2^-3] as expected
```

Now consider the model from the New Zealand "Modeling Measles" paper (part A).

$$\frac{dS_1}{dt}=\mu_0 N_0-\left(\mu_1+\omega(t)\beta\sum_{j=1}^{8}C_{1j}I_j\right)S_1$$

$$\frac{dI_1}{dt}=\omega(t)\beta S_1\sum_{j=1}^{8}C_{1j}I_j-(\mu+\gamma)I_1$$

$$\frac{dS_i}{dt}=\nu_{i-1}S_{i-1}-\left(\mu_i+\omega(t)\beta\sum_{j=1}^{8}C_{ij}I_j\right)S_i,\quad i=2,3,\ldots,8$$

$$\frac{dI_i}{dt}=\mu_{i-1}I_{i-1}+\omega(t)\beta S_i\sum_{j=1}^{8}C_{ij}I_j-(\mu+\gamma)I_i\quad i=2,3,\ldots,8$$

Notice that $\sum_{j=1}^{8}C_{ij}I_j=\left(C\vec{I}\right)_i$ , so the sums give the i[th] entry of the product (which is a vector) of a matrix and a vector. This suggests a convenient way to implement the sums in our code.

```
function xdot = f(x,t)
        #x must be [S;I] or [S,I]
        #f(x,t) returns [dS/dt,dI/dt]
        global mu;      #a vector
        global N0;
        global beta;
        global C;       #a matrix
        global gamma;
        global nu;      #a vector

        n = rows(x)/2;          #if x is a column vector, n is the number of age classes
        if n==.5
        n=columns(x)/2;         #if x is a row vector, n is now the number of age classes
        endif

        for i=1:n
                S(i)=x(i);
                I(i)=x(i+8);
        endfor                  #now we have vector S and vector I seperately
        CIproduct=C*I;          #this is why it's really nice to have I defined and not just x ( =[S;I] )

        xdot(1)=mu(0)*N0-(mu(1)+omega(t)*beta*CIproduct(1))*S(1)
        xdot(1+n)=omega*beta*S(1)*CIproduct(1)-(mu(1)+gamma)*I(1)
        for i=2:n
                xdot(i)=nu(i-1)*S(i-1)-(mu(i)+omega(t)*beta*CIproduct(i))*S(i);
                xdot(i+n)=mu(i-1)*I(i-1)+omega(t)*beta*S(i)*CIproduct(i)-(mu(i)+gamma)*I(i)
        endfor
endfunction
```

I haven't implemented the entire New Zealand model yet, but I see no reason why the code above would not work. It would be nice if we could return dS/dt and dI/dt seperately, but "lsode" requires that the function return just one vector, so the most logical solution is to return [dS/dt;dI/dt] as I have done above.