

MTBI Computer Lab Manual

Introduction to Lab Manual

This book will detail the essentials for the types of modeling one can do using Matlab, and optionally Maple, at MTBI. This book will teach you how to write well documented code, an essential for group collaborations. Techniques covered in the manual for first year students include basics of Matlab commands, numerical integration, construction of bifurcation plots and cobweb diagrams, and the use of packages such as *pplane* and *odesolve*. There are two labs that the first years can attempt to undertake, but may be more appropriate for returning students: simulating the birth-death process as well as a stochastic approach to the SIR model. There are optional labs in the end which cover the basics of and numerical integration using Maple.

The M-files are documented by the first commented block of the file. This part of the file is displayed in the Command window when the `help filename` command is used. It is very important that you understand what each line is doing, especially the input and outputs of the function and the *for* loop that iterates the desired function. When you write your own code for your projects, you will put in comments such as these for each line. (In Matlab, everything following a % on the same line is ignored and thus marks a comment.) An example of documentation given at the beginning of an M-file is such as:

```
% Function:  SDESIR
% Date of Revision:  6.14.07 % Inputs:  beta- rate of infection
% gamma- rate of recovery
%Output:  t- time series of solutions
% I- time series of infective individuals
% Purpose:  This function will evaluate a stochastic instance of the SIR model
using
% Weiner processes.
% Example:  [t,I]=SDESIR(2,.5);
```

It sometimes help to write a pseudo-code solution to your problem, this does not involve writing any code but instead writing down the essence of the algorithm you are trying to execute. For instance:

Step 1- Set up parameters.

Step 2- Initialize the state variables.

Step 3- Within a loop set each successive value of the state variables.

Step 4- Once this is done plot the value of each variable against the independent variable (time, iteration, etc...)

Within this manual something done in **TrueType** denotes a built in command within Matlab and something in *italics* denotes something that can either be entered verbatim or have you specific variable/ function name placed there. Just remember you all have a mathematical outlook to your studies and writing programs is no different, trust yourself and don't be afraid to play around to see what things do.

MTBI New Student Computer Lab

Introduction to Matlab

The objective of this lab is to familiarize you with the Matlab. Carefully read every problem and make sure you understand it. Familiarize yourself with the following commands by either typing `help command` or by simply trying to use them.

General:

`help command` - specify a command and the info about it will be given
`lookfor topic` - specify a topic and all commands that have related topics will be listed
`who` - lists current variables
`cd mtbi` - changes the working directory to mtbi (assuming it exists)
`pwd` - tells you which directory you are currently working in
`dir` - lists files in that directory
`clear all` - clears variables from memory
`format long` - lets you see more significant digits
`size(x)` - gives the size of the variable (matrix) x
`length(x)` - gives the length of the vector x
`zeros(n,m)` - an n x m matrix of zeros
`ones(n,m)` - an n x m matrix of ones
`eye(n,m)` - the n x m identity matrix
`rand(n,m)` - an n x m matrix whose entries are random numbers uniformly distributed in (0,1)
`randn(n,m)` - an n x m matrix whose entries are random numbers normally distributed with mean 0 and variance 1
`linspace(x1,x2,n)` - a linearly spaced vector whose beginning value is x1 and the last value is x2 with a total of n points
`help .` - specifically look at Arithmetic Operators.

Plots:

`figure` - creates a new graph window
`subplot(x,y,z)` - breaks a figure window into x by y separate plot windows where z is the particular window you wish to plot in.

Various line types, plot symbols and colors may be obtained with `plot(x,y,S)` where S is a character string made from one element from any or all the following 3 columns:

y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	- -	dashed
g	green	s	square	p	pentagram
w	white	d	diamond	h	hexagram
k	black	v	triangle (down)	<	triangle (left)
b	blue	^	triangle (up)	>	triangle (right)

For example, `plot(x,y,'c+:')` plots a cyan dotted line with a plus at each data point; `plot(x,y,'bd')` plots blue diamond at each data point but does not draw any line.

Exercise 1: The Intersection of x^2 and x^3 **Write the following script into an .m-file and after each percent sign provide a comment to say what that line of code does.**

```
x=-10:.05:10; %
y=x.^2;%
subplot(2,2,1),plot(x,y);%
xlabel('x');%
ylabel('x^2');%
y1=x.^3;%
subplot(2,2,2),plot(x,y1);%
xlabel('x');%
ylabel('x^3');%
subplot(2,2,3:4),plot(x,y,'r');%
hold on, plot(x,y1,'g');%
plot(0,0,'b*');%
xlabel('x'),ylabel('y');%
legend('x^2','x^3',0);%
title('x^2 vs. x^3');%
axis([-8 8 -100 100]);%
```

Exercise 2: Plot the curves (on the same graph)

$$y = x$$
$$y = r x (1 - x)$$

Find the intersections in the first quadrant (for $r = .5, 1.2, 2.0$) accurate to 6 decimal places (use both `zoom` and `fzero` and explain any difference between the two answers). Use different colors and symbols to plot the curves. Plot all the pictures on the same figure in different subfigures (using `subplot`) and also on the same figure (using `hold on`). Use the command `axis` to make sure the display window is okay. Use the command `legend` to label the graphs. **Provide a picture of the figure (save as a .fig file).**

Pseudocode answer:

- create a vector that has the x-axis coordinates
- define the curves to be plotted
- use subplot to graph the curves in the arrangement you want
- adjust axes for better viewing
- label the curves
- find the intersection points using your eyeball!
- create a script .m-file that will be a function (the top line will be `function retval=zerofinder(x)`)
- This script file will find the difference of the two equations, this is not tricky
- use `fzero(@zerofinder, guess)` in the command window to find the zero near your eyeball's guess.

Exercise 3: Generate n random numbers from both a normal distribution and a uniform distribution for $n = 100, 1000, 5000$. Plot the results in a histogram: **Plot the three outputs using rand on the same figure but different axes using subplot. Do the same with the output of randn. How do the subplots compare?**

Pseudocode answer:

- define variables that call `rand` and `randn` for each sample size
- plot the output using `hist`

MTBI New Student Computer Lab

Numerical Integration in Matlab

In this lab we show how to numerically integrate systems of differential equations using Matlab.

You will be creating two M-files for each problem. The first will be a function which will be called by the Matlab function `ode45`. In this function, you will enter your equations. The exact code you need to enter is given below. The second M-file will contain the code that will tell Matlab to integrate the equations and then plot them. See Example 1.

Example 1: Consider the competition model of rabbits vs. sheep given by

$$\begin{aligned}\frac{dx}{dt} &= x(3 - x - 2y) \\ \frac{dy}{dt} &= y(2 - x - y)\end{aligned}$$

Numerically integrate the differential equations. Plot the solution curves in the x-y plane and draw the vector field, using the same plotting scale as the Maple pictures from class.

Answer:

Create an M-file called `compete.m`:

```
function dx=compete(t,x)

%dx is a vector with entries being the right-hand-sides
%of the system of differential equations
%x in our system is x(1) in Matlab
%y in our system is x(2) in Matlab

dx=[ x(1).*(3-x(1)-2*x(2)); x(2).*(2-x(1)-x(2))];

*****
```

Then create another M-file, name it whatever you wish, entering only the first set of commands. The second set of commands will plot only one solution curve (sometimes this is desirable). Make sure you understand all the lines of both codes.

```
*****NEW m-FILE*****

% to plot multiple trajectories (i.e., solution curves) on the same screen:
t0=0;
tf=25;
tspan=[t0 tf];
x0=[.2 1 .2 4 4 4 4; .4 4 .3 1 4 3 2.5];

%a matrix with I.C.'s (.2,.4), (1,4), etc.

N=length(x0(1,:)); % # of I.C.'s = # of columns of x0
figure
hold on
```

```

for i=1:N
    [t,x]=ode45('compete',tspan,x0(:,i));
    plot(x(:,1),x(:,2));

end;
xlabel('x');
ylabel('y');
title('Competition between rabbits and sheep')
[u,v]=meshgrid(0:.3:4,0:.3:4);
pu=u.*(3-u-2*v);
pv=v.*(2-u-v);
quiver(u,v,pu,pv);
hold off
axis([0 4 0 4])
*****

This is the Code for only one Trajectory
*****

%for plotting one trajectory (solution curve)
t0=0;
tf=25;
tspan=[t0 tf];
x0=[0.2;0.4];
[t,x]=ode45('compete',tspan,x0);

%ode45 is a built-in Matlab function that numerically
%integrates the system of differential equations.
%It returns a vector t and a matrix x whose columns
%contain the corresponding solution of the system at
%those particular t values.

plot(x(:,1),x(:,2));    %this plots x vs y, ie, x(1) vs x(2)
hold on
xlabel('x');
ylabel('y');
title('Competition between rabbits and sheep')
[u,v]=meshgrid(0:.3:4,0:.3:4);
pu=u.*(3-u-2*v);      %these lines are the original equations
pv=v.*(2-u-v);        %and will be used to plot the vector field
quiver(u,v,pu,pv);    %this lines plots the vector field (of arrows)
hold off

Then in the Matlab command window type axis([0 3 0 2]) change the axes to observe
how this changes what you see.

```

Exercise 1: Consider the predator-prey system from class:

$$\begin{aligned}\frac{dx}{dt} &= (3 - y) x \\ \frac{dy}{dt} &= \left(-5 + \frac{x}{2}\right) y\end{aligned}$$

Use Matlab to draw the phase portrait with the vector field, using the command `axis([0 26 0 10])` as the window in which you view the trajectories.

Pseudocode answer: Besides cosmetic changes (the name of the file, title, I.C.'s, etc.), you will need to change 3 lines in the above code. The first change will be in the M-file previously called `compete.m` - you enter your new set of equations there. The other change is in the two lines `pu` and `pv` which generate the vector field.

Exercise 2: Numerically integrate the Lorenz equations

$$\begin{aligned}\frac{dx}{dt} &= \rho (y - x) \\ \frac{dy}{dt} &= x (r - z) - y \\ \frac{dz}{dt} &= x y - b z\end{aligned}$$

for $b = 8/3$, $\rho = 10$, $r = 24.6, 24.9, 28.0$. **Do not try to use the quiver command as you did in the 2-dimensional case above.**

Pseudocode answer:

-Create an m-file called `lorenz.m` with first line *function dx=lorenz(t,x)*

-Enter the parameter values in the m-file

-Enter the right-hand side of the equations as

`dx=[p*(x(2)-x(1)); -x(1).*x(3)+r*x(1)-x(2); x(1).*x(2)-b*x(3)];`

where `x(1):=x`, `x(2):=y`, and `x(3):=z`

-create another m-file that called `lorenz.m`

-give the initial conditions (type `help ode45` and read the first paragraph for the syntax)

-plot the solutions in the `t vs. x` plane and also in 3-D using `plot3`

-label the axes

-use the command `view` to change the orientation of the picture

BONUS: Interesting looks at Lorenz! The plots of the Lorenz attractor all hold a lot of information in and of themselves. By just thinking about dependency `x` depends on `y,z` and `t`, so it would be useful to see how these relationships play out in the attractor. **Try to find a way to plot the `x v. y`, `x v. z`, `x v. t`, `y v. z`, `y v. t`, and `z v. t` pictures all in the same figure window. Also see if you can figure out how to make a “movie” of the path.**

MTBI New Student Computer Lab

Cobwebbing and Bifurcation Diagrams in Matlab

In this lab we will investigate both cobweb and bifurcation plots. This is the first truly difficult lab, but stick with it.

Exercise 1:

a) Create the M-files `cobweb.m` and `logistic.m` by entering the programs at the end of this handout. Run the program `cobweb.m` and make sure all the bugs are worked out. An example of how to call it is given just above the function declaration.

b) Modify the code so that the iterate sequence, n vs. x_n , is also plotted. **Display the cobweb and the iterate sequence on the same figure but not on the same plot (i.e., use subplot).** Note that these changes can be done with the addition of 2-3 lines - the previously entered code does not need to be changed. **Choose values of r between 0 and 4 and observe the output. Which method of displaying the iterates is easiest to observe? Which display is easiest to observe if the number of iterations is increased or decreased? Write down 5 values of r (for $N = 50$) that generate "interesting" outputs.**

c) The pictures displayed all depend on the initial value x_0 . To find the long-term behavior of the system, we ignore a large number of iterates at the beginning. To see the benefit of this, change the above code as follows: you will display 4 pictures (using `subplot(221)`, `...`, `subplot(224)`); the top two pictures will be the exact same as those from the above code; the bottom two pictures will be the cobweb and iteration sequence but with the first 300 iterations ignored (you will need to add the words `subplot(221)` and `subplot(222)` in the appropriate places; besides that, you will need to add exactly two lines which are modifications of two existing lines; remember commands like `x(301:N)` and do not change any of the existing lines); make the call to `cobweb` with $x_0 = 0.4$ and $N = 500$. **Do this for $r = 3.2, r = 3.5$ and again for $r = 3.83$. How important is it to ignore transients when studying the long term behavior?**

Exercise 2: This exercise allows us to explore what happens as we vary r in a systematic way. The display will be called a "bifurcation diagram." For a particular value of r , we will plot the x -values that occur after the transient is ignored. Create a Matlab function called `bifn.m` which has the following properties:

- it takes the name of a function to be iterated, an initial parameter value `ri`, a final value `rf`, an initial guess `x0`, and the number of r -values at which we want to iterate the difference equation
- it varies the parameter r between `ri` and `rf` in equal steps and calls `cobweb.m` (not plotting the cobweb output) at each r -value - note that you will need a for loop to do this
- it ignores the transient values of $x(n)$ and plots r vs. $x(n)$. For $r = 3.2, 3.5, 3.83$, **how do the plots obtained in exercise 1 relate?**

Pseudocode answer:

- create a figure and type hold on
- define the number of iterations to be done (500)
- use `linspace` to divide up the parameter between `ri` and `rf`
- use a for loop to call `cobweb` (which will compute the iterations)
- when you call `cobweb`, do not display the cobweb diagram
- plot the iterates 301 through 500 in the vector `x`, that is, call `cobweb` using 500 iterations and assume that the transient will be gone after the first 300 iterations; in plotting this, Matlab will run faster if you plot a vector vs. a vector (rather than a point vs. a vector); thus, outside of the for loop, define `o=ones(1,200)` and generate the plot inside the for loop by typing `plot(r(j)*o,x(301:N),'.')`

The following two M-files are for exercise 1.

```
% cobweb.m
%
% This function iterates a function f and plots the iterates and the
% corresponding cobweb diagram. The function call has the form:
%
% x = cobweb(f,r,x0,N,flag)
%
% Input:
% -----
% f - function to iterate, this is a string
% r - parameter for f
% x0 - initial point of iteration
% N - number of iterations
% flag - output is displayed when flag is not 0
%
% Output:
% -----
% x = sequence of iterates, length N+1
%
% Examples:
% -----
% x = cobweb('logistic',3.6,0.1,50,1);
% x = cobweb('ricker',10,1,50,0);

function x = cobweb(f,r,x0,N,flag)

%---
% allocate space for iterates - if you know the number of iterations,
% it saves computer resources if you allocate space at the beginning
%---

x = zeros(1,N+1);
%---
% compute the iterates
%---
```

```

x(1) = x0;

for i = 2:N+1
    x(i) = feval(f,r,x(i-1));
end

%---
% create graphical output
%---

if (flag)
    %---
    % display cobweb diagram
    %---
    % open figure and set hold on
    figure;
    hold on;

    % create points of evaluation
    z = linspace(min([0, floor(min(x))]),ceil(max(x)));

    % evaluate production function
    y = feval(f,r,z);

    % plot production function and identity
    plot(z,y,'r');
    plot(z,z,'g');

    % plot cobweb
    stairs([x(1) x(1:N-1)], [0 x(2:N)]);

    %label plot
    title([upper(f) ': Iterates 0 to ' num2str(N) ...
          ' with r = ' num2str(r) ' and x0 = ' num2str(x0)]);

    xlabel('x(n)');
    ylabel('x(n+1)');
end % if

***** Now create another M-file *****

% logistic.m
%
% This function corresponds to the logistic population model equation.
% The function call has the form:
%
% xt = logistic(r,x)
%
% The output xt is the population at the next generation starting from
% x in the model with parameter r.

```

```
function xt = logistic(r,x0)
xt = r*x0.*(1 - x0);
```

MTBI New Student Computer Lab

2-D differential equations using *pplane6* in Matlab

There is a free download that allows Matlab users to easily plot 2-D direction fields and solutions in the phase plane. (You can go to www.google.com and search under *pplane*.) This file should already exist in your default Matlab directory. Type `cd pplane6` and then type `pplane6`. A new window should pop up.

Example 1: The first example is the same competition example as in the previous lab. Enter these equations:

$$\begin{aligned}x' &= x * (3 - x - 2 * y) \\y' &= y * (2 - x - y).\end{aligned}$$

In the display window, set minimum $x = 0$, maximum $x = 3$, minimum $y = 0$, and maximum $y = 2$.

In the direction field box, make sure *arrows* is marked. Click *proceed* and observe the direction field. Click once to trace a trajectory forward and backward in time. Repeat a few times.

Now go to *solutions* \rightarrow *find an equilibrium point*. Then click with mouse where you expect to see an equilibrium. Observe the jacobian is given as are the corresponding eigenvalues and eigenvectors. (You can click *display the linearization* to see what happens near the fixed point.)

In the display window, graph t vs. x and t vs. y by going to *graph* \rightarrow *both*. Click on a trajectory that you want to see plotted.

Now replot the picture but with *lines*, *nullclines*, and *none* checked (three separate plots). Experiment with *number of field points per row or column* to observe how this changes the display window.

Exercise 1: The purpose of this exercise is to use *pplane6* to observe the bifurcations that occur. Consider the equations

$$\begin{aligned}x' &= ax - y + xy^2 \\y' &= bx + y + y^3.\end{aligned}$$

For $a = -0.5, b = 1$, use *pplane6* to find the unique equilibrium point and classify its stability. Draw the phase portrait.

We know from lecture that varying parameters may result in a bifurcation. Suppose we are told that a $\lambda = 0$ bifurcation (i.e., saddle-node, pitchfork, or transcritical) occurs along the line $a = -b$ and that a Hopf bifurcation occurs along the line $a = -1$ for $a > -b$ (i.e., to the right of the $a = b$ line). By using *pplane6*, determine what type of $\lambda = 0$ bifurcation occurs. **How many equilibria were born as this curve was crossed? Then determine whether the Hopf bifurcation that occurs is supercritical or subcritical.**

Pseudocode:

-Draw the two bifurcation curves by hand (in the parameter plane).

-Choose pairs of parameter values that lie close to the bifurcation curves in each of the respective regions and draw a phase portrait with equilibrium points.

-In the case of the Hopf bifurcation, observe whether the limit cycle that is born is stable or unstable. You may need to change the direction that the solutions are plotted (in the display window, go to *options* \rightarrow *solution direction* \rightarrow *forward*). In addition, you might change the time interval over which the solution is plotted (in the display window, go to *solutions* \rightarrow *keyboard input*). On the window that pops up, estimate with the initial condition and also give the time interval over which to integrate.

MTBI Advanced Student Computer Lab

Basic Birth/ Death Process in 0, 1, and 2 Explicit Spatial Dimensions

In this lab we show how to construct a simulation of the basic birth/ death contact process utilizing Matlab.

Exercise 1: The Basic Stochastic Birth/ Death Process.

To start off we will simulate a birth/ death process using the very basic concepts. Birth events will happen with a rate of ϕ births per unit time, and death events with rate μ deaths per unit time. Since this will be a Poisson process only one event may happen at a given time (a birth or a death), with any given event being a birth with probability

$$P(\text{birth}) = \frac{\phi}{\phi + \mu}.$$

Write fully documented code that will simulate this process and find parameter combinations of ϕ and μ that cause extinction and other that cause persistence. Recall that this process is stochastic so the same parameter values may give wildly different results. So it is important to run a simulation several times with the same parameters to determine the “average” behavior. Also initial populations should be relatively high, low populations have the tendency to be driven to extinction much too early. **Record the parameter values that caused extinction at least 4 out of 5 times, ϕ_e, μ_e and record those that cause persistence with an overall population average of about 100, ϕ_p, μ_p .**

Pseudo Code-

1. Write an m-file which is a function, the inputs will include the birth rate, death rate, and initial population, the outputs should be a time series and a population data.
2. Make a for loop which will generate a uniform random variable and decide if the event is a birth or a death.
- 3.a) if it is a Birth then increase the current population by one.
- 3.b) else it is a death so decrease the current population by one.
- 3.c) For each event generate an exponentially distributed random variable with parameter $\frac{1}{\phi + \mu}$.
4. If the population ever hits zero then break; , otherwise allow the for loop to run for 1000 events.
5. Plot the time series data for 5 runs for the persistent values and also plot their average on the same axis, make sure to label things clearly with xlabel, ylabel, and legend.
6. Repeat 5. for the extinction values.

Exercise 2: Construct a 1-Dimensional Stochastic Birth/ Death Simulation.

In this exercise the species in question will exist on a one dimensional space (like a vector). It can be thought of as a row of trays where each tray can hold exactly one individual. The probability of an event being a birth is slightly different and is given by:

$$P(\text{birth}) = \frac{\phi p_c}{\phi p_c + \mu N}$$

where p_c is the current population and N is the total number of available sites. This gives us the correct probability because we are allowing a death event to target any site, while the birth event may only target an occupied site (who in turn will occupy one of his neighbors at random). This setup allows for failed birth and death events. That is the targeted site of the birth event may randomly choose an already inhabited site, in which case nothing happens. Further a death event may target an empty site, this again results in no change. In order to simulate this we will construct an $N \times 1$ array of zeros which shall act as the landscape for each generation we will initialize a certain percentage of this landscape to be inhabited, with values 1,2,3,..., and then run the basic contact process. With each iteration the current generation vector is stored in a matrix which shall hold all of the generation and then visualized via the **image** command. Furthermore, once the simulation has completed it will display a plot of time vs. population to observe. This is a tall order, so below is the majority of the code. **Your task is to comment the code provided and then fill in the code needed. Also as before you should find a pair of values where the population consistently survives and others where the population goes to extinction. How do these compare to the “0-D” simulation? Plot the time series data for 5 simulations along with the average for each pair of parameters.**

```

%Start this m-file with a comment block which will have a brief description
of what this program does.
function poplist=OneDSto(bins,phi,mu,popdensity)
x=zeros(bins,1);
popdensity=.5;%Initial population density
population=floor(popdensity*bins);
cpop=0;%current population
maxiter=50000;
livecoor=zeros(1,1);%This is a technical thing to store where the live sites
are on the landscape
livematrix=zeros(bins,maxiter);%We will use this to draw the simulation
poplist=zeros(1,1);%This will track the population after every event.
neighbor=[-1;1];%Another technical variable used to find our left or right neighbor.

while cpop < population
    r=unidrnd(N);%Produce a discrete uniform random number from 1 to N
    if ~ x(r)%the ~ is to logically negate what follows it
        cpop=cpop+1;
        x(r)=cpop;
        livecoor(cpop)=r;
    end
end

poplist(1)=cpop;
time(1)=0;

for i=1:maxiter
    result='persistence';
    bp=cpop*phi/(cpop*phi+bins*mu);%Compute the birth probability
    time(i+1)=exprnd(1/(phi+mu),1,1)+time(i);%Since this is a Poisson Process
    %we generate an exponential random variable with 1/rate for a parameter.
    livematrix(:,i)=x;
    r=rand(1);
    if(r<bp)%Is it a birth event?
        rcoor=unidrnd(cpop);%generate an integer from 1 to current population
        spot=livecoor(rcoor);
        coin=rand(1,1);%Decide on left or right.

        if(coin<.5)
            new=1;
        else
            new=2;
        end
    end
end

```

```

newspot=spot+neighbor(new);
if(newspot==0)
    newspot=bins;
else if(newspot==bins+1)
    newspot=1;
end
end

%*****
%*****
%**Code the Actual Birth Event Here**
%*****
%*****
else
    rcoor=unidrnd(N);

    if (x(rcoor)>0)
        remove=x(rcoor);
        %to maintain list we shall replace the chosen site on the
        %livecoor list with the last element.
        livecoor(remove)=livecoor(cpop);
        livecoor(cpop)=0;%set last element to zero to keep it tidy
        x(rcoor)=0;%kill the chosen site
        cpop=cpop-1;%mark the death
        if(cpop>0 & remove =cpop+1)
            x(livecoor(remove))=remove;
        end
    end
end
colormap([0 0 0;1 0 0]);
if(mod(i,100)==0)
    image((livematrix(:,1:i)+1)');
    drawnow;
end
poplist(i+1)=cpop;
if(cpop==0)
    result='Extinction';
    break;
end
end
%*****
%*****
%**Write the code to Display population graph**
%*****
%*****

```

Exercise 3: 2-Dimensionalization.

Believe it or not making a 2-dimensional version of the last program is quite simple. Instead of an $n \times 1$ array for a landscape we will have an $n \times n$ matrix. There are a few other technical/ cosmetic changes. **Write down what kind of changes you think should be made. Do you expect the species to have a harder or easier time to survive with same parameters as the 1-dimensional simulation? What do you expect to be the most difficult aspect to program? the easiest? and why?**

Exercise 4: The Simulation.

Below is a listing of the 2-dimensional system code. **Please comment each line and note in the header what has changed from the 1-D to the 2-D simulation. Again find parameters that promote existence and others for extinction. Plot the time series data on a single plot for each case along with their respective averages.**

```
N=20;
x=zeros(N,N);
phi=2;
mu=1;
popdensity=.8;
population=floor(.3*N^2);
cpop=0;
livecoor=zeros(1,2);
while cpop < population
    r=unidrnd(N^2);
    if ~ x(r)
        cpop=cpop+1;
        x(r)=cpop;
        [i,j]=ind2sub(size(x),r);
        livecoor(cpop,1)=i;
        livecoor(cpop,2)=j;
    end
end
maxiter=50000;
poplist=zeros(1,1);
poplist(1)=cpop;
neighbor=[0,-1;0,1;-1,0;1,0];
time(1)=0;
```

```

for i=1:maxiter
    result='persistence';
    bp=cpop*phi/(cpop*phi+N^2*mu);
    time(i+1)=expnrd(1/(cpop*phi+N^2*mu),1,1)+time(i);
    r=rand(1);
    if(r<bp)
        rcoor=unidrnd(cpop);
        spotrow=livecoor(rcoor,1);
        spotcol=livecoor(rcoor,2);
        coin=rand(1,1);

        if(coin<.25)
            new=1;
        elseif(coin<.5)
            new=2;
        elseif(coin<.75)
            new=3;
        else
            new=4;
        end
        newspotrow=spotrow+neighbor(new,1);
        newspotcol=spotcol+neighbor(new,2);
        if(newspotrow==0)
            newspotrow=N;
        else if(newspotrow==N+1)
            newspotrow=1;
        end
        end
        if(newspotcol==0)
            newspotcol=N;
        else if(newspotcol==N+1)
            newspotcol=1;
        end
        end
        index=sub2ind(size(x),newspotrow,newspotcol);
        if (x(index)==0)
            cpop=cpop+1;
            x(index)=cpop;
            livecoor(cpop,1)=newspotrow;
            livecoor(cpop,2)=newspotcol;
        end
    end
end

```

```

else
    rcoor=unidrnd(N^2);
    temp=ind2sub(size(x),rcoor);
    if (x(rcoor)>0)
        remove=x(rcoor);
        livecoor(remove,:)=livecoor(cpop,:);
        %to maintain list we shall replace the chosen site on the livecoor list with the last element.
        livecoor(cpop,:)=livecoor(end,:);%set last element to zero to keep it tidy
        x(rcoor)=0;%kill the chosen site
        cpop=cpop-1;%mark the death
        if(cpop>0 & remove~=cpop+1)
            tempr=livecoor(remove,1);
            tempc=livecoor(remove,2);
            temp=sub2ind(size(x),tempr,tempc);
            x(temp)=remove;
        end
    end
end
end
colormap([0 0 0;1 0 0]);
if(mod(i,1)==0)
    image((x+1)'); drawnow;
end

poplist(i+1)=cpop;
if(cpop==0)
    result='Extinction';
    break;
end
end
end
hold on;
xlabel('Population Distribution');
ylabel('Generation');
title(['2-Dimensional Population Model: 'result]); figure;
temp=length(poplist);
plot(time(1:temp),poplist,'r')

```

MTBI Advanced Student Computer Lab

Stochastic SIR

In this lab we will be producing a program which shall simulate a stochastic SIR process. Two separate methods will be employed in the construction of the programs: continuous time Markov chains and Weiner processes. The former shall be accompanied with a way to calculate the maximum epidemic size, while the latter will have a comparison between the SDE and the ODE model (stochastic and deterministic).

Exercise 1: Stochastic SIR Markov Process.

We will be modeling the SIR epidemic equations:

$$\begin{aligned}\frac{dS}{dt} &= -\beta S \frac{I}{N} \\ \frac{dI}{dt} &= \beta S \frac{I}{N} - \gamma I \\ \frac{dR}{dt} &= \gamma I.\end{aligned}\tag{1}$$

However, due to the lack of birth and death we are dealing with constant population so the R dynamics may be ignored reducing Equation (1) to:

$$\begin{aligned}\frac{dS}{dt} &= -\beta S \frac{I}{N} \\ \frac{dI}{dt} &= \beta S \frac{I}{N} - \gamma I.\end{aligned}\tag{2}$$

This assumption may be valid for modeling a short outbreak of the flu, where the population will remain relatively constant and anyone born into the system is considered quarantined, newborns protected from germs due to behavioral patterns of parents. We know that $\beta S \frac{I}{N}$ is the rate a susceptible person becomes infected and that γI is the rate an infected person is removed from the system, recovered. Thus this may be modelled very similarly to a birth/ death process (“births” and “deaths” into infection). As such this will be modelled as a continuous time Markov chain, Poisson process, where an event is either an infection or a recovery. Recall that the probability of an event being an infection is given by

$$P(\text{Infection}) = \frac{\beta S \frac{I}{N}}{\beta S \frac{I}{N} + \gamma I} = \frac{\frac{\beta}{N} S}{\frac{\beta}{N} S + \gamma}.$$

Therefore write a Matlab script which will simulate this SIR model 3 times and plot each result on the same plot via stairs. What do you notice? Change the script to run the simulation 10, 50, 100, and 500 times without plotting the result but saving each value and then plot the averages. What do you notice now? Estimate the maximum number of infected individuals over the course

of the epidemic.

Pseudocode Answer-

- 1.This will work exactly like the stochastic birth/ death process where the events are either an *infection*, with rate $\beta S \frac{I}{N}$, or a *recovery*, with rate γI .
- 2.When an *infection* occurs the I will increment and the S will decrement.
- 3.When a *recovery* occurs the I will decrement and the S will stay the same.

We may take some time now to calculate the maximum number of infected individuals over the course of the epidemic analytically. **First show that:**

$$\frac{dI}{dS} = \frac{\frac{dI}{dt}}{\frac{dS}{dt}} = -1 + \frac{\gamma N}{\beta S}. \quad (3)$$

Now integrating Equation (3) results in

$$\begin{aligned} \int_0^\tau dI &= \int_0^\tau -1 + \frac{\gamma N}{\beta S} dS \\ I(\tau) - I(0) &= -S(\tau) + \frac{\gamma N}{\beta} \log(S(\tau)) + S(0) - \frac{\gamma N}{\beta} \log(S(0)) \\ I(\tau) &= I(0) - S(\tau) + \frac{\gamma N}{\beta} \log(S(\tau)) + S(0) - \frac{\gamma N}{\beta} \log(S(0)). \end{aligned} \quad (4)$$

With τ being the time the maximum is achieved. But when does this occur? Solving $\frac{dI}{dt} = 0$ results in $S = \frac{\gamma N}{\beta}$, which will be when the graph of $I(t)$ has a critical point, in this case a maximum. Thus we may rewrite Equation (4) for clarity as

$$I_{max} = I_0 - \frac{\gamma N}{\beta} + \frac{\gamma N}{\beta} \log\left(\frac{\gamma N}{\beta}\right) + S_0 - \frac{\gamma N}{\beta} \log(S_0). \quad (5)$$

Generate the average plot of 500 trial runs for $\beta = 2, \gamma = 1, I_0 = 1$ and $S_0 = 99$; for $\beta = 2, \gamma = 1, I_0 = 10$ and $S_0 = 90$; for $\beta = 2, \gamma = 2, I_0 = 1$, and $S_0 = 99$; and lastly for $\beta = .5, \gamma = 1, I_0 = 1$, and $S_0 = 99$ and compare with the result from Equation (5). Which value change seemed to have the largest impact on the maximum number of infected individuals? On the duration of the epidemic?

Exercise 2: Stochastic SIR Weiner Process.

Now we will model this process via using so-called stochastic differential equations, SDE's. This may be thought of simply as adding noise to our differential equations. We must be careful when introducing the noise, to not uncouple like reactions. The system to be simulated is

$$\begin{aligned} dX_1 &= -\frac{\beta}{N} X_1 X_2 dt + B_{11} dW_1 + B_{12} dW_2 \\ dX_2 &= \left(\frac{\beta}{N} X_1 X_2 - \gamma X_2 \right) dt + B_{21} dW_1 + B_{22} dW_2 \end{aligned}$$

where $X_1(t)$ is the number of susceptibles at time t and $X_2(t)$ that of the infected, and each are random variables with $X_i \in [0, N - X_j], i \neq j$. The B_{ij} terms are from a matrix

in the form

$$B = \frac{1}{D} \begin{pmatrix} C_{11} + G & C_{12} \\ C_{21} & C_{22} + G \end{pmatrix}$$

where C is the covariance matrix, $D = \sqrt{C_{11} + C_{22} + 2G}$ and $G = \frac{\beta\gamma X_1 X_2^2}{N}$. These quantities are not very easy to calculate, but for this model they are:

$$C = \begin{pmatrix} \beta X_1 X_2 / N & -\beta X_1 X_2 / N \\ -\beta X_1 X_2 / N & \beta X_1 X_2 / N + \gamma X_2 \end{pmatrix}$$

$$D = \sqrt{\gamma X_2 + 2\beta X_1 X_2 / N + 2\sqrt{\beta\gamma X_1 X_2^2 / N}}$$

Below is the majority of the code which will run this simulation. Note that I wasn't very careful with my notation with the below code. The value x is X_1 and y is X_2 . Make changes if you find them necessary. You are to fill in the blanks and make sure to fully document the code. Solutions can be done just using Forward Euler. Use the fact that the Forward Euler solution of $dX = \mu(X)dt + \sigma(X)dW$ is given by $X(t+dt) = X(t) + \mu(X(t))dt + \sigma(X(t))N(0,1)\sqrt{dt}$, where $N(0,1)$ is a standard-Normal distributed random variable. Compare 3 trials to the deterministic model. What do you notice? Compare 5, 10, 50, and 100. What do you notice now? Take the average of 100 runs and then take the absolute value of the point-wise difference of the average and the deterministic solution. How do they compare?

```

beta=2;
gamma=0.5;
N=100;% Population
T=10;
k=100;
dt=T/k;
for j=1:100% Number of random simulations
    y(1)=50;
    x(1)=N-y(1);
    for i=1:k
        if y(i)<=0
            y(i+1)=0;
            x(i+1)=x(i);
            continue;
        elseif y(i)>=N
            y(i+1)=N;
            x(i+1)=0;
            continue;
        else
            C=[beta*x(i)*y(i)/N, -beta*x(i)*y(i)/N;...
              -beta*x(i)*y(i)/N, beta*x(i)*y(i)/N+gamma*y(i)];
            G=sqrt(beta*gamma*x(i)*y(i)^2/N);
            D=sqrt(C(1,1)+C(2,2)+2*G);
            B=1/D*(C+G*eye(2));
            W1=randn;
            W2=randn;
            %*****
            %Calculate the actual equations!
            %*****
        end
    end
    stairs([0:dt:T],y,'r');
    hold on;
end
S(1)=x(1);
I(1)=y(1);
for i=1:k
    %*****
    %Insert Deterministic Calculations Here.
    %*****
end
plot([0:dt:T],I,'k','LineWidth',2);

```

MTBI Optional New Student Computer Lab

Introduction to Maple

The objective of this lab is to familiarize you with the Maple. Carefully read every problem and make sure you understand it.

General:

Maple has a very good help system. You will find that you will use this feature extensively. Some commands that you should be aware of are: `simplify`, `expand`, `evalf`, `evalc`, `allvalues`, `eliminate`, `coeff`, `diff`, `Diff`, `exp`, `fsolve`, `lhs`, `rhs`, `plot`, `roots`, `solve`, `subs`, `sqrt`, `combine`, `mtaylor`.

This list is not exhaustive but the commands frequently come up in computations. See the help menu for instructions on using them. Maple allows you type text which it ignores in two ways: 1) you may click on the 'T' at the top of the window and then enter the text; 2) you may place the text after the # sign.

A few comments before beginning:

- always start your sessions by typing `restart` on the first line
- all lines must either end with `;` or `:` (the latter is used to suppress output)
- You can make corrections on the same line, you don't have to retype the entire thing
- make sure you give each line you enter a name, e.g., `eq1`, `eq2`, etc. This will be useful if you refer to this line later (see example below).

Example 1:

```
restart; #We always want to begin our sessions this way.
eq1:=sin(x)+3*x^4;
eq2:=Diff(eq1,x);
eq3:=diff(eq1,x);
eq4:=diff(eq1,t);
eq5:=Diff(x,t)=r*x*(1-x);
eq6:=solve(rhs(eq5),x);
eq6a:=eq6[2];
eq7:=Int(rhs(eq5),x);
eq8:=int(rhs(eq5),x);
eq9:=subs(r=2,eq8);
plot(eq9,x=-2..3);
```

Exercise 1: Consider the logistic differential equation with constant effort harvesting

$$\frac{dx}{dt} = r x (1 - x) - \frac{1}{2}x$$

where r is a parameter between 0 and 2. **Find the fixed points (equilibria) of this system as a function of r . For what values of r will there be two nonnegative**

equilibria? Determine the stability of the equilibria analytically (using Maple) and also graphically (using Maple) for the case when there are two positive equilibria. You may need to use the command `simplify`, among others, to simplify the resulting expressions. **For the case $r = \sqrt{2}$, find the value of x at equilibria (both an exact value and a numerical approximation).**

Pseudocode answer:

- enter the equation
- solve that equation (set equal to 0) for x
- take the derivative of the right hand side (rhs) of the original equation
- substitute in the equilibria to this expression
- graph the right hand side of the original equation for various r values
- substitute in $r = \sqrt{2}$ and again use solve
- use evalf to obtain a numeric answer

Exercise 2: Consider the equation

$$\frac{dx}{dt} = \frac{2x^m}{1+x^4} - x$$

where x is a population (i.e., nonnegative). **Determine the number and stability of the real (as opposed to complex) equilibria for $m = 1$ and $m = 2$ both graphically and analytically.**

Pseudocode answer:

- enter the equation
- for $m = 1$, solve the equation (set equal to 0) for the equilibria
- note:** if you do not substitute in a specific value for m , you will not be able to solve the equation
- take the derivative of the right hand side of the equation
- substitute in the equilibria
- graph the original equation
- repeat above steps for $m = 2$

Exercise 3: In class you learned that linearization is equivalent to performing a Taylor expansion and ignoring quadratic and higher terms. This was justified by the fact that quadratic and higher terms are very small (and thus are negligible) if we expand near the equilibria. **Using the Maple command `mtaylor`, calculate the Taylor expansion of exercise 2 above (for $m = 1$ and $m = 2$), keeping up to cubic terms. How does `mtaylor` differ from the command `taylor`? Now keep terms to order 18 (that is, so that the powers of each factor add up to 18 or less). What's happening?**

Note: You will need to type the command `readlib(mtaylor):` in order to use the command `mtaylor`.

MTBI Optional New Student Computer Lab

Numerical Integration in Maple

In this lab we show how to numerically integrate systems of differential equations using Maple. We will also use the trace and determinant of the jacobian matrix to determine stability of the equilibria in our systems.

Example 1: Consider the competition model of rabbits vs. sheep given by

$$\begin{aligned}\frac{dx}{dt} &= x(3 - x - 2y) \\ \frac{dy}{dt} &= y(2 - x - y)\end{aligned}$$

Numerically integrate the differential equations. Plot the solution curves in the x-y plane and draw the vector field.

Answer:

```
restart;
eqx:=x*(3-x-2*y);
eqy:=y*(2-x-y);
sol:=solve(eqx,eqy,x,y); #Be careful! Maple does not always give you the
#solutions in the same order each time you run it.
#The remainder of the code assumes that the output was in the form
#sol := {x = 0, y = 0}, {x = 3, y = 0}, {x = 0, y = 2}, {x = 1, y = 1}
with(linalg):
A:=jacobian([eqx,eqy],[x,y]);
A00:=subs(sol[1],evalm(A));
ev00:=trA00=trace(A00),detA00=det(A00);
#The output shows that we have an unstable node at (0,0).
A30:=subs(sol[2],evalm(A));
ev30:=trA30=trace(A30),detA30=det(A30);
#The output shows that we have a stable node at (3,0).
A02:=subs(sol[3],evalm(A));
ev02:=trA02=trace(A02),detA02=det(A02);
#The output shows that we have a stable node at (0,2);
A11:=subs(sol[4],evalm(A));
ev11:=trA11=trace(A11),detA11=det(A11);
#The output shows that we have a saddle point at (1,1).
#We will calculate the eigenvectors since they give the directions of the
#stable and unstable manifold for the saddle.
evs1:=eigenvecs(A11);
evalf(evs1);
with(DEtools):
eq1a:=diff(x(t),t)=subs(x=x(t),y=y(t),eqx);
eq1b:=diff(y(t),t)=subs(x=x(t),y=y(t),eqy);
IC1:=[[x(0)=.2,y(0)=.4],[x(0)=1,y(0)=4],[x(0)=.2,y(0)=.3],[x(0)=4,y(0)=1],
[x(0)=4,y(0)=4],[x(0)=4,y(0)=3],[x(0)=4,y(0)=2.5]];
```

```
DEplot([eq1a,eq1b],[x(t),y(t)],t=0..25,IC1,x=0..4,y=0..4,scene=[x,y],
stepsize=.05,title='Competition of rabbit vs. sheep',linecolor=black);
```

Exercise 1: Consider the predator-prey system from class:

$$\begin{aligned}\frac{dx}{dt} &= (3 - y) x \\ \frac{dy}{dt} &= \left(-5 + \frac{x}{2}\right) y\end{aligned}$$

Determine the equilibria and their stability. Use Maple to draw the phase portrait with the vector field, using the initial conditions $(x(0), y(0)) = (1, 3), (3, 3), (5, 3), (7, 3), (9, 3)$. Adjust the axes so that $x \in [0, 25], y \in [0, 10]$.

Pseudocode answer:

-Comparing with the above example, you will need to change the equations that you input and your initial conditions.

-Use `thickness=0` if the solutions seem too thick.

Exercise 2: Numerically integrate the Lorenz equations

$$\begin{aligned}\frac{dx}{dt} &= \rho (y - x) \\ \frac{dy}{dt} &= x (r - z) - y \\ \frac{dz}{dt} &= x y - b z\end{aligned}$$

for $b = 8/3, \rho = 10, r = 24.6, 24.9, 28.0$. Do *not* draw the arrows in the vector field (as you did in the 2-dimensional example). **Integrate for $t \in [0, 40]$ and use the initial condition $x(0)=10, y(0)=10, z(0)=-10$. Find the equilibria of the system at these parameter values and determine their stability. If there is time find the equilibria of the system for general b, ρ, r values and draw various plots in 3-dimensional space.**

Pseudocode answer:

-Comparing with the above example, you will need to change the equations that you input and your initial conditions.

-The command `'arrows=none'` will allow you to not draw the arrows.

-Use the command `DEplot3d` to plot the system in 3-dimensions.

-You can left-click on the picture and then hold the left-click and rotate the picture by moving the mouse.