May 11, 2022 – Hands-On Intro to the Basics of Scientific Programming
Prof. Joshua Weitz, Dr. Bradford Taylor, and QBioS Cohort

Would you like to be able to build a computational model of a living system but don't know how? Then you are in the right spot.

This tutorial is meant to be interactive... that is *you* should be reading, typing, thinking, and asking questions. There are no source files to share, the point is for you to think about and enter the code and see the result. And then, to vary the code and see what changes. This is a more effective strategy than just *copying* code others have written and running it.

The materials are adapted from a semester long course entitled "Foundations of Quantitative Biosciences" developed by Prof. Joshua Weitz in Fall 2016/2017/2018 as the cornerstone class of the QBioS Ph.D. at Georgia Tech. The materials have been adapted to focus on stochastic models and to account for the greater variety of backgrounds of students in this week's workshop.

Today we will focus on the basics of coding that can help you build models... whether of gene expression, cellular dynamics, game theory, or some other problem linked to dynamics of living systems. A few notes before we get started:

- If you are experienced in coding, then please work on challenge problems identified in this tutorial.
- Please choose one language - MATLAB or Python - and stick with it for the two day workshop.
- Please work on your own computer, save files for future reference, but also - work with a partner so you can discuss and help each other as you go.

Let's get started!

1 Getting Started

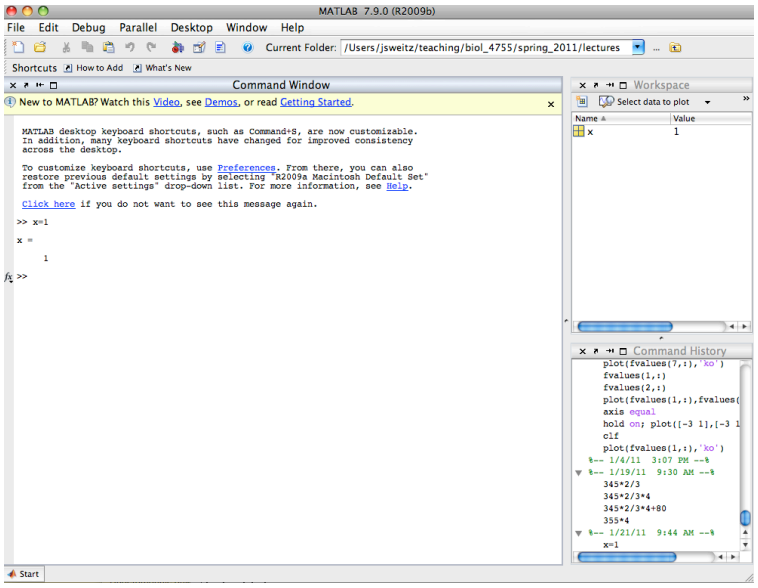
MATLAB is a numerical computing environment and is also a programming language. We will be using it in this class to simulate and model the dynamics of living systems from scales of molecules to ecosystems. MATLAB is particularly good at:

- Matrix manipulation
- Numerical methods
- Algorithms
- Graphical output

and is okay at:

- Symbolic computation
- Interfacing with other languages
- Speed of for loops (as compared to C)

It can also be used for interactive work using the MATLAB Live Editor (similar to the iPython notebook). So, let's get started! In today's class, you will gain practical experiencing using Matlab. When you open Matlab, you should see a set of windows that looks like this:



The key elements are the Command Window, Workspace and Command History. The command window is where you enter commands, and you should see a prompt that looks like this:

```
>>
```

You can do basic math at this prompt, for example

```
>> 3+4
```

In which case the command prompt will return this

```
ans =
     7
```

This means the answer is 7. In the examples that follow, please enter the code following the >> symbols, noting that the answers you receive should match those that follow, e.g.,:

```
>> exp(1)
ans =
    2.7183
```

Matlab has **many** functions that are built in (and you can use it like a calculator), for example, to learn more about the exponential function:

```
>> help exp
 EXP    Exponential.
    EXP(X) is the exponential of the elements of X, e to the X.
    For complex Z=X+i*Y, EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y)).

    See also expm1, log, log10, expm, expint.

    Overloaded methods:
       lti/exp
       codistributed/exp
       fints/exp

    Reference page in Help browser
       doc exp
```

Many functions have names that you expect (how do you think you should calculate cosine or sine of a value, for example)? Try it out!

Matlab is not just a calculator. It is also a programming language that can store values in memory. For example, the command

```
>> x=3
x =
     3
```

tells Matlab that the variable x has the value 3 and now every time you use "x", Matlab will substitute the value 3, for example:

```
>> y=x+1
y =
     4
```

It is very important to realize the "=" sign does not mean that Matlab checks to see if the two sides are equal to each other, but rather states that whatever is on the left should be assigned the value of that on the right. If you want to check the truth of a particular statement, for example is x equal to 3, or alternatively, is y equal to 3 then you would type:

```
>> x==3
ans =
     1
>> y==3
ans =
     0
```

The double "==" sign tells Matlab to logically compare what is on the left with that on the right and return 1 if true and 0 if false. Note that if you don't want Matlab to report back the answer every time, you can use the semicolon

```
>> y=x+1;
```

Matlab can also handle arrays of values (for example a vector or a matrix). The simplest way is to use the colon, which defines a sequential vector, for example

```
>> v=1:5
v =
     1     2     3     4     5
```

you can also modify the increments by adding a step in the middle

```
>> w=1:2:9
w =
     1     3     5     7     9
```

Any entry can be examined using the parentheses

```
>> w(3)
ans =
     5
```

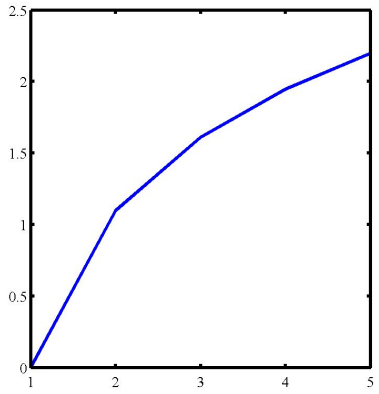
and basic math can be performed automatically on vectors (and matrices), for example

```
>> log(2*w)
ans =
    0.6931    1.7918    2.3026    2.6391    2.8904
```

Matlab can also plot graphs and surfaces and all sorts of things. To create a simple plot, use the plot command

```
>> plot(v,log(w))
```

which leads to:



2 Building "Programs" from "Scripts" and "Functions"

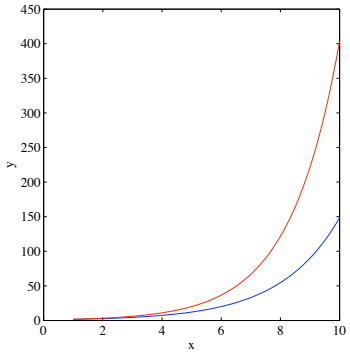
However, once you have a lot of commands, it will get exhausting typing them (especially when you make mistakes). So, Matlab has a concept of a script, which is a list of commands in a file that you can execute directly from the command window. To create a script go to the File menu and select New > M-file. You can also start a new file using any other editor. Please note that in your files, you can include comments with the % symbol, these are not executed by MATLAB and are for your reference online. Now write a few commands, such as:

```
% my_first_file.m
% Create some vectors
x =1:0.1:10;
y1 =exp(0.5*x);
y2 =exp(0.6*x);
% Plot the vectors
figure  %Pullupanewfigure
hold on %Allow multiple plots on the same figure
plot(x,y1)
plot(x,y2,'r') % Use a red line
xlabel('x'), ylabel('y')  %Label the axes
print -dpdf my_first.pdf % Save the image to a file
```

Save this file as my_first_file.m in the same folder you're currently working in. Now go to the Matlab prompt and type

```
>> my_first_file
```

This should execute all the commands in your script, yielding your first figure.



If it does not, you may have to change your directory so that you run the scripts from the same directory, or add this path to your directory listing.

The problem with this script is that if you wanted to change the exponents in these files, you would have to edit the script and then re-run it, instead of designating a variable change from the command window. Moreover, we cannot have a script return a variable or accept a variable as input. To exchange variables between code blocks, we need to use a concept called a "function". Functions are program files that can be called from the Command window - open a new M-file and type:

```
function dNdt=logGrowth(t,N)
% function dNdt=logGrowth(t,N)
%
% logGrowth gives the growth rate of a population of size N at time t
% this is also the name of the function to be called in MATLAB
% Usage:dNdt=logGrowth(t,N)
r =0.5;
K =100;
dNdt =r*N.*(1-N/K);
```

Now you can use your function like one of Matlab's, for example:

```
>> vN =0:110;
>> figure
>> plot(vN, logGrowth(0,vN))
>> xlabel('N'),ylabel('dN/dt')
```

gives an upside down parabola, denoting that growth rate is positive between 0 and 100 and negative when N is greater than 100.

3 Getting Started with Core Techniques

Note that in the following sections, we will omit the command line signal `>>`*. These commands can be entered directly on the command line, or as part of scripts and functions.*

3.1 Create loops

The two types of loops we will discuss are `for` and `while` loops. Both loops start with a keyword such as `for` or `while` and they end with the word `end`. The `for` loop allows us to repeat certain commands many times with a "counter" variable. Here is one example:

```
for j=1:4 % counting
  j+2 % the statement you want to repeat
end
```

You can also increment your counter variable by any real number like

```
for j=1:-0.1:0 % counting
  1-j % the statement you want to repeat
end
```

Can you figure out what this means given the output? Or you can even use a set of arbitrary values:

```
for j=[1 3 5 7]
  j-1 % the statement you want to repeat
end
```

Challenge problem: Exercise on matrices

Define a random matrix A of size 3-by-3. Use a double `for` loop to calculate the square of the entries in A and store the values in another matrix B. (Hint: type "help rand" in Matlab if you don't know how to define a random matrix)

The second type of loop is the `while` loop. The `while` loop repeats a sequence of commands as long as some condition is met. For example, given a number n, the following code will return the smallest non-negative integer a such that $2^a \geq n$. This code should be saved in its own file, named `smallexp.m`:

```
function a = smallexp(n)

a = 0;
while 2^a < n
  a = a + 1; % statement to execute if the condition is met
end
```

Now in the command window, try this out:

```
>> a = smallexp(4)

a =
     2
```

Note that in the above example we used the conditional statement, $2^a < n$ to decide whether the statement within the while loop should be proceeded. Such conditional statements are also used in "if" statements that are discussed in the next section. The rational operator used here is "<", which means "less than." Other rational operators that are available in Matlab include:

```
>  greater than
<= less than or equal
>= greater than or equal
== equal
~= not equal
```

Simple conditional statements can be combined by logical operators

```
(&&, ||, ~)
```

into compound expressing such as the following:

```
(5 > 1) && (5 == 6)
```

3.2 Make decisions

Now, let's suppose you want your code to make a decision, and the "if" statement is what you need. The general form in Matlab is as follows:

```
if expression1
  statements1
elseif expression2
  statements2
else
  statements3
end
```

Challenge problem: Recursive factorials

Exercise: Finish the following pseudo-code that gives the factorial of a positive number n, using the recursive formula $n! = n(n-1)$. Replace the ... with your code.

```
function N = factorial_recur(n)
% function N = factorial_recur(n)
% Recursive implementation of the factorial

  if ...
      N = 1;
  else
      N = ...
  end
end
```

3.3 Go fast, i.e., "vectorize"

Remember: `for` loops are SLOW in Matlab. One way to make your Matlab run faster is to "vectorize" the algorithm you use in the code. Vectorization can be done by converting `for` and `while` loops to equivalent vector or matrix operations. A simple example would be the following (`tic` and `toc` are used as a stop watch). You can enter this in a script or on the command window.

```
x = 1;
tic
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
toc

tic
x = .01:.01:10;
y = log10(x);
toc
```

For more complicated code, vectorization is not always so obvious. Some of the most commonly used functions for vectorizing can be found in the Help browser of Matlab. Nonetheless, there is a simple procedure that will help: elementwise operators.

3.4 Elementwise operators

In MATLAB, you can do calculations element by element. For example, if you have a variable `X` and want to square it, you might think to write `X^2` where the ^ sign denotes raising `X` to a certain power. Let's see what happens

```
X = 1:10;
X^2
```

Did it work? No! The reason is that MATLAB interprets `X` as a matrix, which you've defined to have 1 row and 10 columns. You cannot multiple a 1 x 10 matrix by a 1 x 10 matrix (this is a discussion for another day). Instead you want to multiply element by element. Let's try again.

```
X = 1:10;
X.^2
```

Did it work? Yes! The dot – . – informs MATLAB to interpret the subsequent operator element by element. This is also helpful for a whole matrix.

```
X = 5*ones(4,4)
X.^2
X^2
```

Contrast the two answers. In the first, you squared all the 5-s. In the second you created a 4x4 matrix filled with the number 5 and then multiplied two square matrices with those elements by each other.

Challenge problem: Element-wise operations: square root

Take the square root of the numbers between 1 to 20 using an elementwise operator.

3.5 Find values you want to know about

Given an array X, the command

```
find(X)
```

returns the indices of all nonzero elements of X. You can also use a logical expression to define X. For example,

```
find(X>2)
```

returns the indices corresponding to the entries of X that are greater than 2. Notice that X could also be a matrix. In this case, the function returns the linear indices as if the matrix was stored as a single column of elements (Try it!). However, when you want to locate the entries that satisfy more than one logical expressions, the 'element-wise' logical operators (& and |) are used in place of 'short-circuit' logical operators (&& and ||).

Challenge problem: Finding entries in matrices

Build a 10-by-10 random matrix A using the command

```
A = rand(10)
```

Then, find the linear indices of entries whose value is smaller than 1/4 and bigger than 1/6. Check the answer with your neighbors.

3.6 Save and load data

The keywords in Matlab are straightforward: save and load. Type

```
save filename
```

and Matlab will save all variables in the current workspace in the file filename. If you do not specify an extension to the filename, Matlab uses .mat. When you want to load all the variables from the file specified by filename, just type

```
load filename
```

Please keep in mind that these file formats are binary, i.e., not human-readable. If you want to save a particular file format use the help to save variables in standard, comma-separated value file format. There are also alternative ways to load and print data.

4 Dynamics

The stochastic model of gene expression that we will study during this workshop is an extension of a deterministic model that consists of ordinary differential equations (ODEs). Some basic concepts from the theory of ODEs will be helpful in understanding how the averages of your stochastic simulations evolve in time, so we cover them here. The term "system" is used here to describe a mathematical model consisting of any number of differential equations.

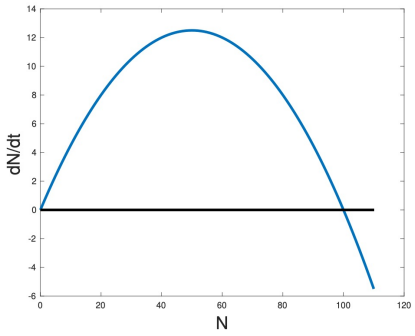
4.1 One-dimensional systems & bistability

A nice example of a one-dimensional ODE is that of logistic growth:

$$\frac{\mathrm{d}N}{\mathrm{d}t} = rN\left(1 - \frac{N}{K}\right)$$

where $N = N(t)$ is a variable describing a population size as a function of time, $r > 0$ is a constant parameter that describes the population growth rate, and $K > 0$ is a constant parameter describing the carrying capacity of the population. It is instructive to plot this equation directly along with the horizontal line representing $\mathrm{d}N/\mathrm{d}t = 0$:

```
vN = 0:110;
r = 1/2;
K = 100;
logGrowth = @(N,r,K) r*N.*(1-N/K);

F = figure;
hold on;
plot(vN, logGrowth(vN,r,K),'LineWidth',3);
plot(vN,zeros(length(vN)),'k','LineWidth',3)
xlabel('N','FontSize',20);
ylabel('dN/dt','FontSize',20);
box on;
hold off;

vN(logGrowth(vN,r,K)==0)
```



The above plot is called the **phase portrait**. From the phase portrait, it is clear graphically that the derivative peaks when the population is equal to half of the carrying capacity $N = K/2$, with a value of $\mathrm{d}N/\mathrm{d}t = rK/4$. The population sizes where the derivative is equal to zero $\mathrm{d}N/\mathrm{d}t = 0$ are of special interest. They are called **fixed points**, let us denote them N^*; they can be found algebraically by setting the above equation to zero and solving for N^*:

$$0 = rN^*\left(1 - \frac{N^*}{K}\right)$$

$$N^* = 0 \text{ or } K$$

The fixed point $N^* = 0$ is **unstable**, since small deviations from it result in a population size that tends away from the fixed point. On the other hand, $N^* = K$ is a **stable** fixed point because small deviations from it result in population sizes that tend back to $N^* = K$. This model is **monostable**, since there is only one stable fixed point. In this model of population growth, the stable fixed point is the carrying capacity K for any choice of parameter values for r and K.

In the following, the words "gene" and "protein" are used interchangeably. Don't get caught up in the biological details. The emphasis here is how the notion of genetic bistability can be understood in terms of mathematical models.

Challenge problem: Stability analysis

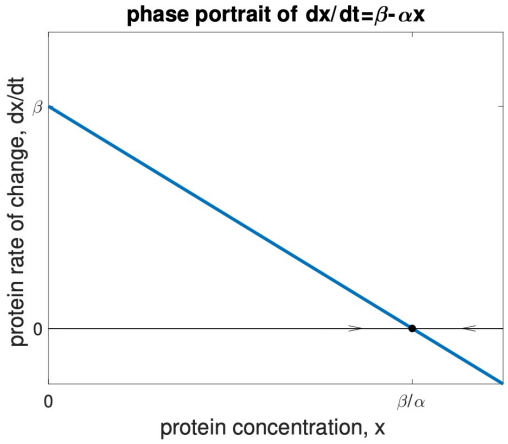
Find the fixed point(s) and their stability for the following 1-dimensional system. Sketch the phase portrait. Is the system monostable?

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \beta - \alpha x$$

where $\beta > 0$ and $\alpha > 0$ are constant parameters representing production and degradation, respectively; $x > 0$ is a variable representing the concentration of protein X. The phase portrait is depicted below, along with the code that produced it. Don't worry too much about understanding the code; it is there only for your reference. The important part to understand is why the phase portrait looks the way it does for this particular system.

```
% define what will be plotted
xVec = linspace(0,5,6);
dxdt = @(x,alpha,beta) beta-alpha*x;
% set parameter values
alpha = 3/4;
beta = 3;
```

```
xfix = xVec(dxdt(xVec,alpha,beta)==0);
% make the plot
figure;
hold on; box on;
plot(xVec, dxdt(xVec,alpha,beta), 'LineWidth',3)
plot(xVec, zeros(length(xVec)),'k')
plot(xfix,0,'ko','MarkerFaceColor','k')
quiver(xfix-1,0,xfix-(xfix-0.5),0,'MaxHeadSize',1,'Color','k')
quiver(xfix+1,0,xfix-(xfix+0.5),0,'MaxHeadSize',1,'Color','k')
% stylize
ylim([-inf dxdt(xVec(1),alpha,beta)+1])
ax=gca;
xticks([0 xfix])
xticklabels({'0','\beta/\alpha'})
yticks([0 dxdt(xVec(1),alpha,beta)]);
yticklabels({'0','\beta'});
ax.FontSize = 16;
title('phase portrait of {dx}/{dt}=\beta-\alphax','FontSize',20)
ylabel('protein rate of change, dx/dt','FontSize',20)
xlabel('protein concentration, x','FontSize',20)
hold off;
```



The problem above demonstrates a simple model of gene expression where protein is produced at a constant rate β and degrades at a rate αx proportional to the instantaneous amount of protein present. The phase portrait for this system is a downward-sloping line that intersects the $\mathrm{d}x/\mathrm{d}t = 0$ axis once at the protein concentration $x = \beta/\alpha$, which is a stable fixed point. Since this is the only stable fixed point for any choice of parameter values for α and β, this system is monostable.

We now study a 1-dimensional model of **bistability**. This bistability requires that protein production become a function of some variable, instead of being a constant rate β, as in exercise 5.1. Suppose that gene X is regulated by its own concentration. We study the system:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) - \alpha x$$

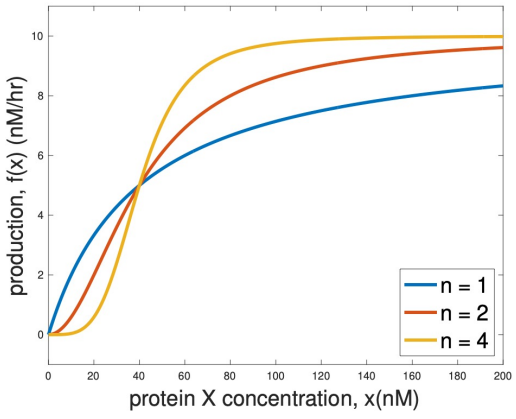
where x is the concentration of protein X and $\alpha > 0$ is the rate of degradation of x. If we constrain $f(x)$ to be monotonic, it will either be a strictly increasing or a strictly decreasing function of x. To start, let's

focus on the increasing case. Consider a production function of the form:

$$f(x) = \frac{\beta x^n}{K^n + x^n}$$

To get an intuition for this function, let us define it and make some plots for various parameter values. For simplicity, let us fix the maximum production rate at $\beta = 10$ nM/hr and the half-activation concentration at $K = 40$ nM, while varying the cooperativity parameter n. Note that all of these parameters are taken to be positive numbers.

```
activator = @(x,beta,K,n) (beta*x.^n)./(K^n+x.^n);

% study the effect of parameter n
xVec = linspace(0,200,400); %defines vector of x values to plot f over
fVec_nEq1 = activator(xVec, 10, 40, 1); % define f(x) with n=1
fVec_nEq2 = activator(xVec, 10, 40, 2); % define f(x) with n=2
fVec_nEq4 = activator(xVec, 10, 40, 4); % define f(x) with n=4

% visualization
figure;
hold on; box on;
plot(xVec, fVec_nEq1,'LineWidth',3); % plot for n=1
plot(xVec, fVec_nEq2,'LineWidth',3); % plot for n=2
plot(xVec, fVec_nEq4,'LineWidth',3); % plot for n=4
ylim([-1 11])
lgd = legend('n = 1','n = 2','n = 4','Location','southeast');
lgd.FontSize = 20;
xlabel('protein X concentration, x(nM)','FontSize',20);% x-axis label
ylabel('production, f(x) (nM/hr)', 'Fontsize', 20); %y-axis label
hold off;
```



From the plot above, it is clear that increasing the cooperativity parameter n yields a production curve that increases at a faster rate with respect to x. This curve is bounded between $f(x) = 0$ and $f(x) = \beta = 10$. The parameter β therefore determines the maximum production rate. At $x = K = 40$, the curve reaches the half-activation point $f(x) = \beta/2 = 10/2 = 5$. So we may call K the half-activation concentration.

Just as we did for the logistic model, let us now analyze $\mathrm{d}x/\mathrm{d}t = f(x) - \alpha x$ with the activator function defined above. The fixed points x^* result when the derivative is zero $\mathrm{d}x/\mathrm{d}t = 0$, which occurs when

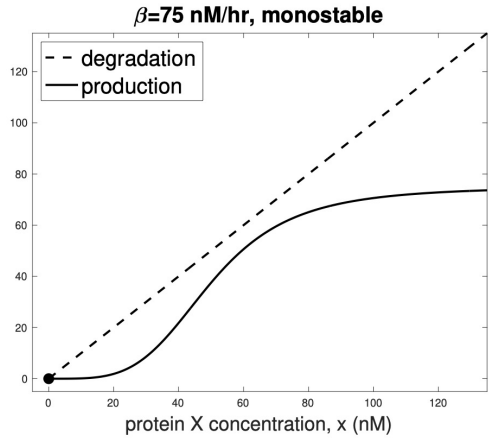
$$0 = f(x^*) - \alpha x^*$$

$$\alpha x^* = f(x^*)$$

$$\alpha x^* = \frac{\beta (x^*)^n}{K^n + (x^*)^n}$$

Although this equation can be solved algebraically for x^*, let us take a graphical approach. We plot the left-hand side of the above equation and the right-hand side separately. The location(s) where these curves intersect yield the solution(s) x^* of the fixed point equation above.

```
% set fixed parameter values
alpha = 1; % 1/hr
n = 4; % dimensionless
K = 50; % nM

% case 1: monostable regime
beta = 75; % nM/hr
xVec = linspace(0,135,400); % define vector of x values to plot over
degr = alpha*xVec; % straight line of slope 'alpha' going through the origin
prod = activator(xVec, beta, K, n); % define the production curve
FPs_x = xVec(degr==prod); % find the fixed points (x-coordinates)
FPs_y = alpha*FPs_x; % y-coordinates of the fixed points

% visualization
figure;
hold on; box on;
plot(xVec, degr, 'k--', 'LineWidth', 2);
plot(xVec, prod, 'k-', 'LineWidth', 2);
plot(FPs_x, FPs_y, 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 9)
xlim([-5 inf]);
ylim([-5 inf]);
lgd = legend('degradation','production','Location','northwest');
lgd.FontSize = 20;
xlabel('protein X concentration, x (nM)', 'fontsize',18) % x-axis label
title('\beta=75 nM/hr, monostable', 'fontsize',20) % slap a title on there
hold off;
```

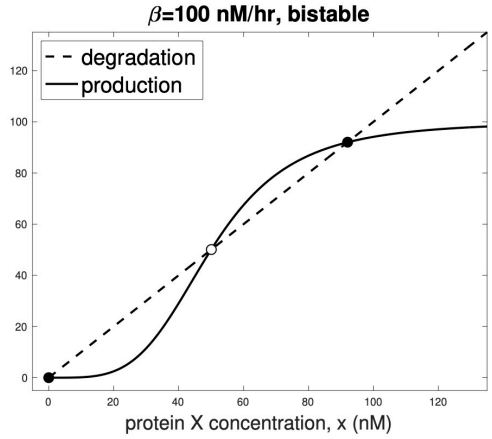


There is a single fixed point at $x = 0$. What is its stability? This can be answered by inspecting the sign of the derivative in a small neighborhood around the fixed point. Since the derivative is given by $\mathrm{d}x/\mathrm{d}t = \text{production} - \text{degradation}$ and $\text{production} - \text{degradation} < 0$ in a small neighborhood to the right of the fixed point in the plot above, the protein concentration x decreases toward this fixed point. Therefore, this fixed point is stable and the system is monostable in this parameter regime with $\beta = 75$.

As the name implies, a **bistable** system is one with two coexisting stable states. How might we render this system to be bistable? Remember that the parameter β sets the maximum of the production curve. So if we increase β from 75, then then production curve shown above will have a higher maximum. If increased enough, that curve will intersect the dashed degradation line while maintaining its previous intersection at the origin. This will create two new fixed points, rendering the system bistable. Let's see this in action, through some code:

```
% case 2: bistable regime
beta = 100; % nM/hr
xVec = linspace(0,135,400); % define vector of x values to plot over
degr = alpha*xVec; % straight line of slope 'alpha' going through the origin
prod = activator(xVec, beta, K, n); % define the production curve
FPs_x = xVec(abs(degr-prod)<0.15); % find the fixed points (x-coordinates)
FPs_y = alpha*FPs_x; % y-coordinates of the fixed points
fprintf('Stable fixed points: (%f, %f)\n',FPs_x(1), FPs_y(1));
fprintf('Stable fixed points: (%f, %f)\n',FPs_x(3), FPs_y(3));
fprintf('Unstable fixed points: (%f, %f)\n',FPs_x([2]), FPs_y([2]));

% visualization
figure;
hold on; box on;
plot(xVec, degr, 'k--', 'LineWidth', 2);
plot(xVec, prod, 'k-', 'LineWidth', 2);
plot(FPs_x([1,3]), FPs_y([1,3]), 'ko', 'MarkerFaceColor', 'k', 'MarkerSize', 9) % plot stable fixed poi
plot(FPs_x([2]), FPs_y([2]), 'ko', 'MarkerFaceColor', 'w', 'MarkerSize', 9) % plot unstable fixed point
xlim([-5 inf]);
ylim([-5 inf]);
lgd = legend('degradation','production','Location','northwest');
lgd.FontSize = 20;
xlabel('protein X concentration, x (nM)', 'fontsize',18) % x-axis label
title('\beta=100 nM/hr, bistable', 'fontsize',20) % slap a title on there
```

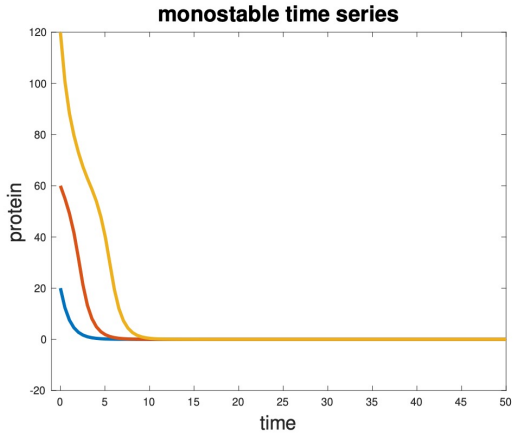
```
hold off;
```

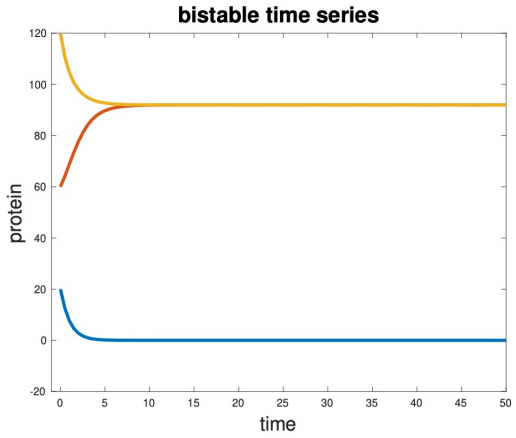


We can use Matlab to numerically solve differential equations, like the logistic growth equation or the protein production equation defined above. We will use a function called 'ode45'.

```
In []:  % solve the protein production equation numerically for two different values of
% beta, showcasing monostability versus bistability.

% Define ODE function
proteinproduction = @(t, x, beta, alpha, K, n) (beta*x.^n)./(K^n+x.^n)-alpha*x;

alpha = 1; % 1/hr
n = 4; % dimensionless
K = 50; % nM

% Numerical solution of the protein production equations
t0 = 0; % Initial time
tf = 50; % Final time
T = linspace(t0, tf); % time steps to report
x0 = [20,60,120]; % Initial protein concentrations

%%%% CHANGE TO 100 TO SEE CASE 2 (BISTABLE) %%%%
beta = 75; % original value is 75
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
hold on; box on;
for j = 1:length(x0)
    [T, vNint] = ode45(@(t,x) proteinproduction(t,x,beta, alpha, K, n), T, x0(j));
    plot(T, vNint, 'LineWidth', 3); % Plot numerically integrated solution
end
xlim([-1 inf])
if beta == 100
    title('bistable time series', 'FontSize', 20);  % change the title for bistable case
else
```

```
    title('monostable time series', 'FontSize', 20);
end
xlabel('time','FontSize', 18);
ylabel('protein','FontSize', 18);
hold off;
```

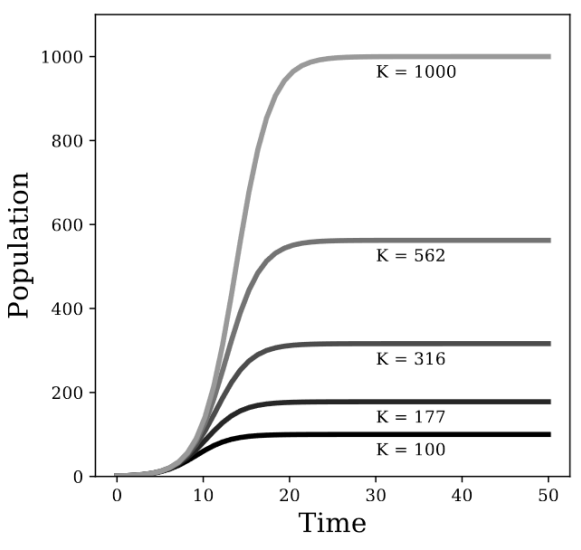




Google matlab ode45 for more information.

Challenge problem: Variables and Differential Equations

Read the 'ode45' documentation and modify your logistic growth model to accept r and K as parameters. Vary the value of K over 1 order of magnitude and compare the results. If your code works, it should look something like this:

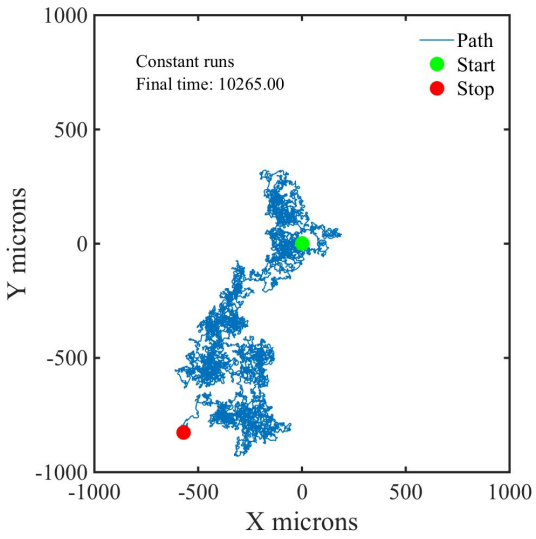


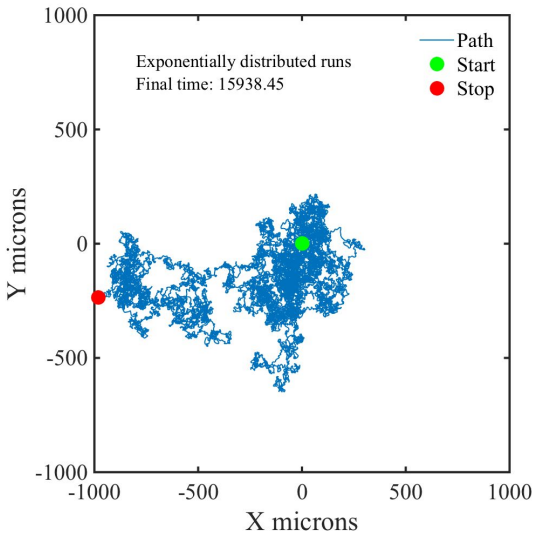
5 Advanced - Individual-Based Simulations

For those more experienced in MATLAB, try and develop a simulation of "diffusion" in which a bacteria swims at a speed of 10 μm/s with each "run" lasting 1 second. In this case the direction of the next run is random. How long will it take, on average, for the bacteria to travel 1mm away from its source? According to the rules of diffusive motion, the average time to travel a distance x should be:

$$T = \frac{x^2}{D} \tag{1}$$

What do we mean by *average* here? If you develop code, can you visualize the results? If you change the length in equation, how does T change? Can you confirm the scaling and estimate D? What happens if the durations of each run is exponentially distributed, so that runs last on average 1 second, but can vary in duration? Did the answer with respect to travel time to reach 1mm change in a quantitative or qualitative way? We get much further into these ideas in the Foundations of Quantitative Biosciences course. Here are examples of two runs, one with constant duration and one with exponentially distributed durations:





6 Solutions to Challenge Problems

This section includes solutions... try to figure this out on your own before you look! Really! You can do it!

Solution to Challenge Problem: Exercise on Matrices. The solution involves using the `rand` command as well as using a pair of for loops to change each element across all combinations of rows and columns:

```
A = rand(3,3);
for i=1:3,
  for j=1:3,
    B(i,j)=A(i,j)^2;
  end
end
```

Solution to Challenge Problem: Recursive Factorials. The solution to this challenge problem involves having a function call itself. This is called a 'recursive' solution. By breaking a big problem into a smaller one, it is possible to generate elegant (but potentially dangerous) solutions. Note that recursive approaches are used in core bioinformatics algorithms, like BLAST.

```
function N = factorial_recur(n)
% function N = factorial_recur(n)
% Recursive implementation of the factorial

  if (n==1)
      N = 1;
  else
      N = n*factorial_recur(n-1);
  end
end
```

Solution to Challenge Problem: Element-wise Operations: Square Root. To take the square root of the numbers between 1 to 20 using an elementwise operator, try out the following, first generating an array and then using the square root operator or exponentiating all elements to the 0.5 power:

```
X = 1:20;
% Option 1
sqrt(X)
% Option 2
X.^0.5
```

Solution to Challenge Problem: Finding Entries in Matrices. The answer to this challenge involves using the `find` command to satisfy two conditions, as follows:

```
A = rand(10);
tmpi=find(A<0.25 & A>(1/6));
```

Note that despite the fact that the matrix is 2D, MATLAB can use linear indexing. To verify your answers, try out the following

```
A(tmpi)
```

If you do this correctly, you'll find that the values are all between 0.1666 and 0.25.

Solution to Challenge Problem: Variables and Differential Equations. Although some variables may be fixed in a given system of equations, there are many circumstances where it is necessary to probe the response of a system to variation in such conditions. In this case, it is possible to pass variables to the ode45 command, which will, in turn, pass these variables on the function specified to actually compute the rate of change in the system of equations. It's best seen by example. First, here is a modified code for intLog.m that generates a structure of parameters called pars and sets up a loop to change the value of K:

```
function dNdt=logGrowth_withpars(t,N,pars)
% function dNdt=logGrowth_withpars(t,N,pars)
%
% logGrowth gives the growth rate of a population of size N at time t
% Usage:dNdt=logGrowth(t,N,pars
% pars is a structure that contains parameters
r = pars.r;
K = pars.K;
dNdt =r*N.*(1-N/K);
```

Next, here is a script that calls this new function multiple times, each with a new value of K and then plots the dynamics on the same axes.

```
% Parameters
t0 =0; % Initial time
tf =50; % Final time
N0 =1; % Initial population size
pars.r=0.5;
Krange =logspace(2,3,5); % Generate 5 logarithmically spaced values

for i=1:length(Krange),
pars.K=Krange(i); % Set the value of K
[T, vNint] = ode45(@logGrowth_withpars, [t0 tf], N0,[],pars); %Numerically integrate
tmph=plot(T,vNint,'k-');
set(tmph,'linewidth',3,'color',[0.75 0.75 0.75]*(i-1)/length(Krange));
hold on
tmpt=text(40,pars.K-30,sprintf('$K=%3.0f$',pars.K),'interpreter','latex');
end
ylim([0 1100]);
```