Quantitative Understanding in Biology Module IV: ODEs Lecture II: Linear ODEs and Stability

Linear Differential Equations

You will recall from the previous lecture that the solution to the canonical ordinary linear differential equation…

$$\frac{dx}{dt} = \lambda x$$

…is…

$$x = ce^{\lambda t}$$

Biological systems that we would be interested in modeling will, of course, usually involve more than one variable. This will result in a system of ordinary differential equations. If we get lucky and this set happens to be a set of linear differential equations, we can apply techniques similar to those we studied for linear difference equations. In general, systems of biological interest will not result in a set of linear ODEs, so don't expect to get lucky too often. However, the analysis of sets of linear ODEs is very useful when considering the stability of non-linear systems at equilibrium. For that reason, we will pursue this avenue of investigation for a little while.

As we did with their difference equation analogs, we will begin by considering a 2x2 system of linear differential equations. The results can be generalized to larger systems. The arbitrary 2x2 system can be written as…

$$\frac{dx_1}{dt} = a_{11}x_1 + a_{12}x_2$$

$$\frac{dx_2}{dt} = a_{21}x_1 + a_{22}x_2$$

This can also be written in matrix form as…

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{Ax}$$

…where **x** is a vector and **A** is a matrix.

In the spirit of our solution to the difference equation problem, we will propose that the solution to our single-variable problem will also work here. We begin by eliminating x_2 from our system. This can be achieved by differentiating the first equation in our set and then rearranging…

$$\frac{d}{dt}\left[\frac{dx_1}{dt}\right] = \frac{d}{dt}a_{11}x_1 + \frac{d}{dt}a_{12}x_2$$

$$\frac{d^2x_1}{dt^2} = a_{11}\frac{dx_1}{dt} + a_{12}\frac{dx_2}{dt}$$

$$\frac{d^2x_1}{dt^2} = a_{11}\frac{dx_1}{dt} + a_{12}[a_{21}x_1 + a_{22}x_2]$$

$$\frac{d^2x_1}{dt^2} = a_{11}\frac{dx_1}{dt} + a_{12}a_{21}x_1 + a_{22}a_{12}x_2$$

The combination $a_{12}x_2$ can be rewritten by looking at the very first equation…

$$a_{12}x_2 = \frac{dx_1}{dt} - a_{11}x_1$$

Using this relation, we can continue…

$$\frac{d^2x_1}{dt^2} = a_{11}\frac{dx_1}{dt} + a_{12}a_{21}x_1 + a_{22}\left[\frac{dx_1}{dt} - a_{11}x_1\right]$$

$$\frac{d^2x_1}{dt^2} = a_{11}\frac{dx_1}{dt} + a_{12}a_{21}x_1 + a_{22}\frac{dx_1}{dt} - a_{11}a_{22}x_1$$

$$\frac{d^2x_1}{dt^2} - (a_{11} + a_{22})\frac{dx_1}{dt} + (a_{11}a_{22} - a_{12}a_{21})x_1 = 0$$

Now we consider our proposed solution. One of the mathemagical properties of the function e^x is that it is its own derivative. That is

$$\frac{d}{dx}e^x = e^x$$

When a coefficient to x appears, the chain rule gives us

$$\frac{d}{dx}e^{\lambda x} = \lambda e^{\lambda x}$$

Since we have a second derivative in our system, we'll also need

$$\frac{d^2}{dx^2}e^{\lambda x} = \lambda^2 e^{\lambda x}$$

So we now can write…

$$\lambda^2 e^{\lambda t} - (a_{11} + a_{22})\lambda e^{\lambda t} + (a_{11}a_{22} - a_{12}a_{21})e^{\lambda t} = 0$$

We can eliminate the $e^{\lambda t}$, and we are left with a quadratic characteristic equation…

$$\lambda^2 - (a_{11} + a_{22})\lambda + (a_{11}a_{22} - a_{12}a_{21}) = 0$$

In fact, this is exactly the same characteristic equation we encountered when we studied systems of linear difference equations. This should make us very happy, because it means that we can skip the rest of the boring algebra and jump right to the solution:

$$\lambda_{1,2} = \frac{\beta \pm \sqrt{\beta^2 - 4\gamma}}{2}$$

$$\beta = a_{11} + a_{22}$$

$$\gamma = a_{11}a_{22} - a_{12}a_{21}$$

$$x_1(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t}$$

$$x_2(t) = d_1 e^{\lambda_1 t} + d_2 e^{\lambda_2 t}$$

Note that the λs are the eigenvalues of the matrix A, just as we are used to. These came from the characteristic equation involving matrix coefficients. Note, however, that the interpretation of the eigenvalues for a differential equation problem is not the same as that of a difference equation problem. Since the eigenvalues appear in expressions of $e^{\lambda t}$, we know that systems will grow when $\lambda>0$ and fizzle when $\lambda<0$.

We encountered eigenvectors in our study of difference equations, and the same ideas apply here. In the above solution, there are four arbitrary constants, c_1, c_2, d_1, and d_2, yet there are only two degrees of freedom (determined by the two initial conditions). It turns out that the ratio c_1/d_1 is always fixed, as is the ratio c_2/d_2. This gives rise to the eigenvectors. We can therefore write our solution neatly in vector form (redefining c_1 and c_2 as we go) as

$$\boldsymbol{x}(t) = c_1 \boldsymbol{v_1} e^{\lambda_1 t} + c_2 \boldsymbol{v_2} e^{\lambda_2 t}$$

Note that **x** is a vector of state variables. $\mathbf{v_1}$ and $\mathbf{v_2}$ are also vectors; the eigenvectors of the matrix **A**.

Complex Eigenvalues Revisited

Since we have a quadratic characteristic equation, we should consider the possibility of complex (and repeated) roots. As usual, we will not consider repeated roots here; consult a text on differential equations if you need to worry about this. The solution we wrote holds for complex eigenvalues; however it is worth considering how to interpret the results.

Everything follows and can be reasoned from the basic identity

$$e^{i\theta} = \cos(\theta) + i \cdot \sin(\theta)$$

If an eigenvalue is complex, we can write…

$$\lambda = r + c \cdot i$$

…where r is the real part and c is the imaginary part. Terms such as $e^{\lambda t}$ now take the form…

$$e^{(r+c \cdot i)t} = e^{rt}e^{i \cdot ct} = e^{rt}[\cos(\mathrm{ct}) + i \cdot \sin(\mathrm{ct})]$$

As with real eigenvalues, the rules for interpreting complex eigenvalues are a bit different when applied to differential equations. We still see that complex eigenvalues yield oscillating solutions. However, we note that the real part of the eigenvalue determines whether the system will grow or shrink in the long term, and the imaginary part determines the frequency.

Expressing the general solution in terms of real parts only involves a fair amount of algebra, which we won't worry about. For completeness, we will simply state that when you have a system with complex eigenvalues (which always come in conjugate pairs), the solution can be written as…

$$x(t) = c_1 e^{rt}[a \cdot \cos(ct) - b \cdot \sin(ct)] + c_2 e^{rt}[a \cdot \sin(ct) - b \cdot \cos(ct)]$$

In this expression, r and c are the real and complex parts of the eigenvalue, and a and b are the real and complex parts of the corresponding component of the eigenvector. Don't worry about the details here, the important part is above; i.e., r gives you long-term growth or decay, and c gives you frequency. Both of these are embedded in the eigenvalue, λ.

Equilibrium Solutions to Dynamic Systems

The above techniques apply to linear dynamic systems. The behavior of non-linear dynamic systems can be quite complex, and in general cannot be treated analytically. You do know, however, how to run numerical simulations of arbitrary dynamic systems using Matlab.

Even though we can't solve for the time-evolution of arbitrary non-linear systems, there are some techniques we can use to help us qualitatively understand their behavior. The first thing one usually does when analyzing a dynamic system is see if there are any steady-states. This is easily done by setting all of the time derivatives to zero, and trying to solve the resultant algebraic equations.

For example, consider again our simple system:

$$\frac{dx}{dt} = \lambda x$$

Setting the derivative to zero yields…

$$\lambda x = 0$$

…which implies x=0. This is something of a trivial solution, as it tells us that we get no growth if we don't have anything to start with. What is important is that there is no other steady-state solution unless λ=0, in which case any value of x is a steady-state solution because nothing ever grows.

Consider a more complex model, the linearized repressilator from the last lab. We had…

$$\frac{dp_A}{dt} = \beta(m_A - p_A)$$

At steady state, we can see that $m_A=p_A$. This is useful, as it allows us to eliminate some variables under steady-state conditions. Note that similar equations would imply $m_B=p_B$, and $m_C=p_C$. We also had…

$$\frac{dm_A}{dt} = \alpha_0 - m_A - \alpha \cdot p_C$$

…which, for steady state conditions, implies…

$$0 = \alpha_0 - m_A - \alpha \cdot m_C$$

Similar reasoning gives…

$$0 = \alpha_0 - m_C - \alpha \cdot m_B$$

$$0 = \alpha_0 - m_B - \alpha \cdot m_A$$

This is a system of three linear equations in three unknowns, and can be readily solved by the normal methods. However, note that there is a symmetry to the problem, which implies that the solution must satisfy $m_A=m_B=m_C$. By inspection we then have…

$$m_A = m_B = m_C = \frac{\alpha_0}{1+\alpha}$$

But wait a minute… When we did the lab exercise, we saw that under some circumstances the system reached a steady state, and in others it did not. Yet the analysis we just did implies that there should always be a steady state solution at the point above.

In the first simulation for our lab, we had $\alpha = 1$; $\alpha_0 = 4$; $\beta = 0.01$. This suggests that we should see a steady state solution at $m_A=m_B=m_C=p_A=p_B=p_C=2$.

We can model the repressilator system in Matlab using differential equations and the ode45 solver. We first create an m-file that returns derivatives:

```matlab
function [ dydt ] = repressilator( t, y, alpha0, alpha, beta )
%REPRESSELATOR Returns derivitaes from the Repressilator model
%   The state variables are:
%   y(1) = ma
%   y(2) = pa
%   y(3) = mb
%   y(4) = pb
%   y(5) = mc
%   y(6) = pc

% Fetch variables
ma = y(1);
pa = y(2);
mb = y(3);
pb = y(4);
mc = y(5);
pc = y(6);
```

```
dydt = [
         alpha0 - ma - alpha * pc,
         beta * (ma - pa),
         alpha0 - mb - alpha * pa,
         beta * (mb - pb),
         alpha0 - mc - alpha * pb,
         beta * (mc - pc),
      ];
```

Then, in Matlab, we can run the model

```
tspan = [0 1000];
start = [ 0, 1, 0, 0, 0, 0];
alpha0 = 4;
beta=0.01;
alpha = 1;
[t, y] = ode45(@repressilator, tspan, start, [], alpha0, alpha, beta);
```

Plots show that we indeed reach a steady state where all concentrations are equal to two.

```
plot(t,y);
```

Introduction to Stability

Now we will run another case, where α=2.5…

```
alpha=2.5;
[t, y] = ode45(@repressilator, tspan, start, [], alpha0, alpha, beta);
plot(t,y);
```

…and we see that the model expands indefinitely. We expected a steady state at

$$m_i = p_i = \frac{\alpha_0}{1+\alpha} = \frac{4}{1+2.5} \cong 1.1429$$

What if we start our simulation near this point…

```
start = [1.1429, 1.1429, 1.1429, 1.1429, 1.1429, 1.1429];
[t, y] = ode45(@repressilator, tspan, start, [], alpha0, alpha, beta);
```

…we see plots that show some interesting patterns.

```
plot(t,y);
plot(y(:,1), y(:,2))
plot(y(:,2), y(:,4))
```

Pay particular attention to the scales of the plots. It seems that if you start at the steady-state solution, you can stay on it; but if you start far away, you won't find it.

There are two possibilities here. The first is that the steady-state solution is not stable. This is analogous to a boulder balanced precariously on the top of a mountain. Leave it be, and it will stay there. But give it a shove (or start your simulation anywhere other than a perfectly balanced boulder), and the simulation will diverge from the equilibrium point. Let's see what happens in our case…

```
start = [1.1429, 1.1429, 1.1429, 1.1429, 1.1429, 1.35];
[t, y] = ode45(@repressilator, tspan, start, [], alpha0, alpha, beta);
plot(t,y(:,2));
plot(y(:,1), y(:,2))
plot(y(:,2), y(:,4))
```

The system spirals away from the equilibrium point. So we can see that the equilibrium is **not** stable.

The 'landscape' of our repressilator system is a bit complex. For example, suppose we started the simulation at

```
start = [1.1, 1.1, 1.1, 1.1, 1.1, 1.1];
[t, y] = ode45(@repressilator, tspan, start, [], alpha0, alpha, beta);
plot(t,y(:,2));
plot(y(:,1), y(:,2))
```

The system appears stable. This is analogous to a walking a ridge or a tightrope. Our boulder, if perfectly balanced on a ridge, can roll down the ridge and find a saddle-like region to come to rest. However, if anything pushes it off the center of the ridge, it will fall off. Unstable equilibria are often saddles, which means that there are some directions in which a perturbation will seems stable, and some in which it will not. An equilibrium point is considered (mathematically) stable if it can survive a push in **all possible** directions.

The second possibility is that the equilibrium is stable, but we did not find a path to it. This is analogous to a mountain range with two valleys. A boulder sitting in one basin is stable (you can kick it in any direction and it will come back to its stable resting point. If your boulder was at the top of a mountain, which basin it falls into will depend on where it started from. The two basins are called attractors (because they attract, or pull, points on a trajectory near them to the equilibrium position). We will see a system like this soon.

The bottom line is that finding equilibrium points is usually not that hard to do analytically; you just set the derivatives to zero. However, checking for stability by experimentation is not so great. We want an analytical means of doing so.

For linear systems, we have the answer already. We look at the eigenvalues of our system. The real parts will tell us if the system explodes or not. Let's try this for our repressilator system. We begin by formulating the matrix that describes our system of differential equations. Note that this is different from the matrix derived from the same system for the difference equation formulation; it must be because that formulation involved choosing a timestep, τ, which we don't need to do here…

The system in matrix form is…

$$\begin{pmatrix} \frac{dm_A}{dt} \\ \frac{dp_A}{dt} \\ \frac{dm_B}{dt} \\ \frac{dp_B}{dt} \\ \frac{dm_C}{dt} \\ \frac{dp_C}{dt} \\ \frac{dy}{dt} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & -\alpha & \alpha_0 \\ \beta & -\beta & 0 & 0 & 0 & 0 & 0 \\ 0 & -\alpha & -1 & 0 & 0 & 0 & -\alpha_0 \\ 0 & 0 & \beta & -\beta & 0 & 0 & 0 \\ 0 & 0 & 0 & -\alpha & -1 & 0 & \alpha_0 \\ 0 & 0 & 0 & 0 & \beta & -\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} m_A \\ p_A \\ m_B \\ p_B \\ m_C \\ p_C \\ 1 \end{pmatrix}$$

We can write a function in Matlab (file is replmat.m) that will give us this matrix for particular values of α_0, α, and β:

```
function [ replmat ] = replmat( alpha0, alpha, beta )
%REPLMAT return a matrix for the repressilator
%   Detailed explanation goes here
replmat = [  -1,      0,     0,       0,     0, -alpha, alpha0;
           beta,  -beta,     0,       0,     0,      0,      0;
              0, -alpha,    -1,       0,     0,      0, alpha0;
              0,      0, beta,   -beta,     0,      0,      0;
              0,      0,     0, -alpha,    -1,      0, alpha0;
              0,      0,     0,       0, beta,  -beta,      0;
              0,      0,     0,       0,     0,      0,      0];
```

Using Matlab, we can quickly compute the eigenvalues of this (linear) system for different values of the parameters. This is similar to what you did in the lab.

```
>> eig(replmat(4.0, 1.0, 0.01))

ans =

  -1.0051 + 0.0087i
  -1.0051 - 0.0087i
  -0.9898
  -0.0202
  -0.0049 + 0.0087i
  -0.0049 - 0.0087i
        0
```

Here we see that the real parts of all eigenvalues are negative, except for the last, which is exactly zero. This implies that the system will reach a stable steady state.

Now let us compute the eigenvalues for the system with α=2.5…

```
>> eig(replmat(4.0, 2.5, 0.01))

ans =

  -1.0129 + 0.0213i
  -1.0129 - 0.0213i
  -0.9741
  -0.0359
   0.0029 + 0.0213i
   0.0029 - 0.0213i
        0
```

Here we see that there is a conjugate pair of eigenvalues with positive real parts; this system is going to 'explode'.

Note also that the imaginary parts of the eigenvales are equal to 0.0213. This tells us something about the frequency of the osciallations we will see. Recall that the solution to our system will have a term involving cos(ct) and sin(ct). These trigonometric functions complete their oscillations in 2π radians, so we expect to see the first oscillation complete when ct=2π. Or, in other words, when t=2π/c≈295. Simple inspection of plots that we previously generated confirms an oscillatory period of about 300.

You can see from the above discussion that the frequency of oscillation is proportional to c, the imaginary part of a number. Or you can view this as the period of oscillation as being proportional to 1/c. Now, if you view a real number as a special case of a complex number where the imaginary part, c, just happens to be zero, then you can view a real eigenvalue as giving rise to a solution with an infinitely long period (or, if you prefer, a frequency of zero).

Linearizing a 1D System

Note that we were able to use eigen analysis to look at stability here because the system was linear. For a non-linear system, this doesn't work directly because we can't represent the system in matrix form, so there is nothing to compute eigenvalues and eigenvectors from.

However, if we are interested in stability around a specific point, we can always approximate the system as a linear model in the region around that point. We can then use eigen analysis on the linearized model to tell us if the system is stable *at that point*. Let's look at a one-dimensional example first…

$$\frac{dx}{dt} = \sin(x)$$

Because sin(x) is periodic, it has infinitely many steady-state solutions. We'll consider just two, the one at x=0, and the one at x=π.

We begin the process of linearization by recalling that a Taylor series expansion of a function, f(x), about a point, a, is…

$$f(x) = f(a) + f'(a) \cdot (x - a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

This can be written more compactly as…

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Note that when the series is infinite (and the function actually has infinitely many derivatives), this is an exact equality, as is written.

When in the neighborhood of x=a, the series may be truncated without much loss of accuracy. Therefore, we can write…

$$f(x) \cong f(a) + f'(a) \cdot (x - a)$$

If we take f(x) to be our original expression for dx/dt, and a to be the equilibrium position of zero, we get…

$$f(x) = \sin(x) \cong \sin(0) + \cos(0) \cdot (x)$$

$$\sin(x) \cong x$$

This is a good approximation to know in general. It merely says that for small angles, sin(x) can be approximated by x itself. Using Matlab, we can see sin(0.01) = 0.0100, and sin(0.1) = 0.0998 (to within four decimal places).

So, in the neighborhood of the equilibrium position, we can approximate our system as the by now very familiar…

$$\frac{dx}{dt} \cong x$$

The eigenvalue of this system is, of course λ=1. This indicates that the equilibrium point is not stable, and that small perturbations from it will cause the system to begin to diverge. Note that we cannot draw any conclusions about the long-term behavior of this non-linear system from this eigenvalue. This is only informative in the neighborhood of x=0 because we used a truncated Taylor series. Once the system starts to diverge, the Taylor series approximation breaks down.

We can follow similar logic to investigate the other steady-state point we wish to consider, that of x=π. At this point, the Taylor series is…

$$f(x) = \sin(x) \cong \sin(\pi) + \cos(\pi) \cdot [x - \pi]$$

$$\sin(x) \cong -[x - \pi]$$

Again, in the neighborhood of our equilibrium position, we can write…

$$\frac{dx}{dt} \cong -[x - \pi]$$

The eigenvalue for this system is, of course, λ=-1. This indicates that the system is stable in the neighborhood of this equilibrium.

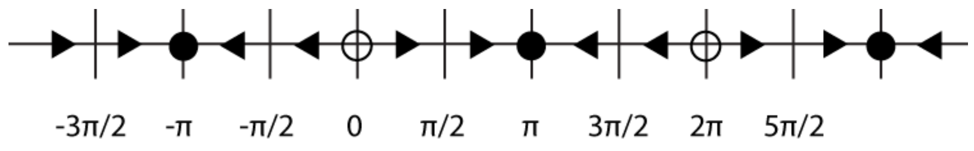
Note that whenever we write a truncated Taylor series around an equilibrium point, the first term is always zero (because it is an equilibrium point). The second term has the form…

$$\frac{d}{dx}\left[\frac{dx}{dt}\right]\Bigg|_{x=a}$$

The work we have done here can be extended to two more dimensions. We will shortly consider a model non-linear system and investigate its stability characteristics.

A 1D Phase Portrait

The stability of the above system can be rationalized in a graphical manner. We can draw a one-dimensional phase portrait of this system as follows.

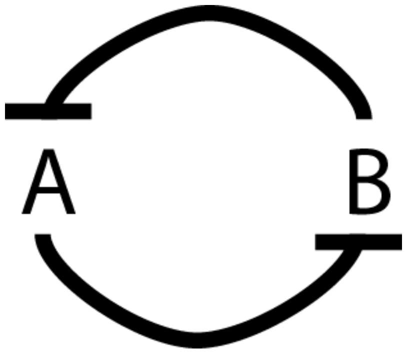


In locations where the value of the derivative is positive, we draw a right-facing arrow to indicate that x will increase. Conversely, we draw left-facing arrows wherever the derivative is negative. The stable and unstable points occur where the derivative changes sign (that is what happens to numbers when they cross zero, after all). Stable points have arrows pointing into them, and unstable points have arrows pointing away.

A Genetic Switch

Reference: Gardner, Cantor, and Collins; *Science*, 2000, as described in Ellner and Guckenheimer, *Dynamic Models in Biology*.

Let us consider a system similar to the repressilator, but with only two components instead of three. We can summarize the system with a diagram like this:



Using reasoning similar to that which we applied to the repressilator, we may be able to anticipate what this system will do. If component A is overexpressed, this will lead to an inhibition of B. The reduction in B will then lead to an increase in A. So we expect that the system, in this case, will favor A over B. However, the opposite case is also a possibility; if B starts out overexpressed, the resultant inhibition of A will lead to a reinforcement of the excess levels of B. Qualitatively, we expect this system to act as a kind of switch, favoring A or B. Just from the diagram it is hard to say anything about stability, that will have to wait for a more quantitative treatment.

In our treatment of this switch model, we will make several assumptions that will make the system easier to study mathematically. While some of them may seem to be a bit of a reach, recall that the purpose of the system here is to serve as a model for studying dynamic systems in general.

If we look back at the repressilator model, we see that we had equations like…

$$\frac{dp_A}{dt} = \beta(m_A - p_A)$$

In the switch model, we will omit such equations and simply take m_i=p_i. This is tantamount to assuming that β is large, so the mRNA and protein quickly reach equilibrium. This is a convenient assumption for us, as it allows us to remove two variables from our system. Since m_A=p_A, we will simple write this as x. Similarly, we take m_B=p_B=y.

You can also think of this process as a biological control system that tries to keep two concentrations equal. The time constant for the controller is proportional to 1/β.

Now when we write the differential equation for mRNA, we'll put a new spin on our model and write…

$$\frac{dx}{dt} = -x + \frac{\alpha_x}{1 + y^{n_B}}$$

The first term represents degradation of A. The second term represents production of A. In the absence of B (i.e., when y=0), production will be at its maximum of α_x. When the system is rich in B, production approaches zero. Note that production of A is non-linear in y; we have borrowed a Hill-like formulation from our study of cooperative enzymatic action.

Let us compare this to the linear equation we used in the repressilator:

$$\frac{dm_A}{dt} = \alpha_0 - m_A - \alpha \cdot p_C$$

In this equation, the third term is meant to represent inhibition of production of the mRNA of gene A. Basal levels of production were given by α_0, and were reduced by the term $\alpha \cdot p_c$. While this is fine when moderate amounts of the inhibitor are present, things get ugly when there is an abundance of it. In this case, inhibition of production can turn into active degradation, which is not what we wanted to model. The non-linear formulation we use for our switch model overcomes this problem.

Another way of looking at this is to realize that the linear model used in our formulation of the repressilator can admit negative values of the concentrations of the components. This is clearly not biologically reasonable. A model that does not allow this to happen is clearly better; here this improvement comes at the non-trivial cost of introducing non-linearity.

We can now repeat our differential equation for A, and write a similar one for B, but add two simplifying assumptions in the process. We will take $\alpha_{x} = \alpha_{y} = \alpha$ and take $n_A = n_B = n$. This makes some of our work easier, although you could model the system just as well without these assumptions.

$$\frac{dx}{dt} = -x + \frac{\alpha}{1 + y^n}$$

$$\frac{dy}{dt} = -y + \frac{\alpha}{1 + x^n}$$

To start our exploration of this system, we will take parameters α=3 and n=2. To begin, we look for steady state points by setting the derivatives of our system to zero. This yields the equations…

$$x = \frac{3}{1 + y^2}$$

$$y = \frac{3}{1 + x^2}$$

It is not obvious how to solve this system of simultaneous non-linear algebraic equations. It is helpful to plot them on the same set of axes …

```
x1 = 0:0.1:10;
y1 = 3 ./ (1 + x1 .^ 2);
y2 = 0:0.1:10;
x2 = 3 ./ (1 + y2 .^ 2);
plot(x1,y1,x2,y2);
```

In this case, we can see that there are three equilibrium solutions.

The curves we plotted are call nullclines. Whenever we are on a nullcline, one of the derivatives is zero. When the nullclines cross, both derivatives are zero and we are at a steady state. Of course, each steady state must be evaluated separately for stability. Since the model is non-linear we cannot simply compute the eigenvalues of our model matrix (there isn't one). However, if we can formulate a linear approximation of the behavior of the model in the region of each steady-state point, we can compute the eigenvectors of the approximate model and draw some conclustions about the stability of the model in that region.

But first, of course, we have to find the intersections of the nullclines. While it might be possible to analytically solve for the intersection of our two nullclines in this case, in general it is not possible to analytically solve a system of arbitrary non-linear algebraic equations. Even if it were possible, it may often be quite difficult to do so. We will therefore be content to solve them numerically, using Matlab to do the heavy lifting. First we define a function that returns the "errors" in our equations (file switch_ss.m):

```
function [ F ] = switch_ss( v )
%SWITCH_SS look nullclines for switch model

x = v(1);
y = v(2);

F = [ x - 3/(1+y^2);
      y - 3/(1+x^2)];
```

Now we can use the Matlab function `fsolve` to solve this system for us. This function is part of Matlab's Optimization Toolbox, which is a licensed add-on to Matlab. Assuming that this is available to you, simply provide `fsolve` with the system you are trying to solve, and an initial guess. Since we have three steady-state points that we need to solve for, we will need to do this three times. In each case we will provide a starting point close to the intersection that we are interested in; the plot we made earlier will inform our initial guesses.[1]

```
x1 = fsolve(@switch_ss, [1.2;1.1])
```

[1] Some versions of Matlab surprisingly don't converge on the solution properly with default parameters for fsolve. This seems to be due to a change in the maximum step size from 0.1 to Infinity. For Matlab 2011a, you may want to use the older default value for this parameter; this can be run with the command…

```
 x1 = fsolve(@switch_ss, [1.2;1.1], optimset('DiffMaxChange',0.1))
```

```
x2 = fsolve(@switch_ss, [0.3;2.6])
x3 = fsolve(@switch_ss, [2.6;0.3])
```

We should verify that these are indeed solutions: `switch_ss(x2)`. Our results are

```
>> [x1 x2 x3]
ans =
    1.2134    0.3820    2.6180
    1.2134    2.6180    0.3820
```

Linearizing a Multidimensional System

Now we want to assess the stability of each of these points. As was done in the 1D case, we will use a truncated Taylor series centered on the steady state point of interest. A truncated 2D Taylor series looks like this:

$$f(x,y) \cong f(x_0,y_0) + (x-x_0)\left.\frac{\partial f}{\partial x}\right|_{x_0,y_0} + (y-y_0)\left.\frac{\partial f}{\partial y}\right|_{x_0,y_0}$$

For the first differential equation in our system, it turns out that the partial derivative with respect to x is fairly trivial…

$$f(x,y) = \frac{dx}{dt} = -x + \frac{\alpha}{1+y^n}$$

$$\frac{\partial f}{\partial x} = -1$$

$$\left.\frac{\partial f}{\partial x}\right|_{x_0,y_0} = -1$$

The partial derivative with respect to y involves recalling a bit of differential calculus…

$$\frac{\partial f}{\partial y} = -\frac{\alpha n y^{n-1}}{(1+y^n)^2}$$

$$\left.\frac{\partial f}{\partial y}\right|_{x_0,y_0} = -\frac{\alpha n y_0^{n-1}}{(1+y_0^{n-1})^2} = -3\cdot 2\cdot\frac{(1.2134)^{2-1}}{(1+1.2134^2)^2} = -1.1911$$

So our linear approximation of the system is…

$$\frac{dx}{dt} = -1\cdot(x-x_0) - 1.1911\cdot(y-y_0)$$

By symmetry, we also have…

$$\frac{dy}{dt} = -1.1911 \cdot (x - x_0) - 1 \cdot (y - y_0)$$

At this point it is useful to make a variable transformation. Let us define $x^*=x-x_0$ and $y^*=y-y_0$. We then have

$$\frac{dx^*}{dt} = -1 \cdot x^* - 1.1911 \cdot y^*$$

$$\frac{dy^*}{dt} = -1.1911 \cdot x^* - 1 \cdot y^*$$

This can be written in matrix form as…

$$\frac{d\boldsymbol{x}^*}{dt} = \boldsymbol{J} \cdot \boldsymbol{x}^*$$

…where J is the matrix…

$$J = \begin{pmatrix} -1 & -1.1911 \\ -1.1911 & -1 \end{pmatrix}$$

Remember, this is an approximation of our model. Around the point ($x=x_0$,$y=y_0$), or alternatively around ($x^*=0$,$y^*=0$), it is a pretty good approximation of the model. So, if we want to know if our first steady-state point is stable, we can compute the eigenvectors of this matrix.

```
>> J = [ -1, -1.1911; -1.1911, -1];
>> eig(J)

ans =

   -2.1911
    0.1911
```

We see a positive eigenvalue, and conclude that this equilibrium is not stable.

Symbolic Math With a Computer

We will take a brief diversion here and explore some of the symbolic math capabilities of Matlab. If figuring out the derivative above gave you pause or brought back bad memories, than this section is for you.

While it is well recognized that computers are quite good at numerical work (such as solving for the intersection of the nullclines), with the right software they also have good symbolic mathematical capabilities. Here we will show you how use Matlab’s Symbolic Math Toolbox (also a licensed add-on) to do some of the heavy lifting.

When working with symbolic math, it is a good idea to make sure you don't have any extraneous variables in your workspace. For current purposes, enter the `clear` command to make sure (save anything you want to keep first).

Next, define which symbols you want Matlab to treat as variables that don't have values. In our case, the expression we want to differentiate uses the symbols x, y, alpha, and n. We enter the following command:

```
>> syms x y alpha n
```

Now we can define a function that we want to differentiate.

```
>> f = -x + alpha/(1+y^n);
```

We can now ask Matlab to differentiate our newly defined function with respect to x…

```
>> diff(f,x)
 ans =
 -1
```

…and with respect to y…

```
 >> diff(f,y)
 ans =
 -alpha/(1+y^n)^2*y^n*n/y
```

Careful inspection of the second result shows that it is equivalent to our result; however, Matlab may need a little encouragement to write the answer a bit more neatly…[2]

```
>> simplify(ans)
ans =
-alpha/(1+y^n)^2*y^(n-1)*n
```

This is just a demonstration; Matlab can do more than this. If you ever need to do a lot of symbolic work like this, you may also want to look at Mathematica.

The Jacobian

We can do the same thing for the second and third steady-state points. But before we do so, let's look back at how we got the matrix J. In general, this matrix, which is known as the Jacobian, will be the matrix…

[2] Later versions of Matlab don't need such encouragement.

$$J = \left.\begin{pmatrix} \frac{\partial}{\partial x}\left(\frac{dx}{dt}\right) & \frac{\partial}{\partial y}\left(\frac{dx}{dt}\right) \\ \frac{\partial}{\partial x}\left(\frac{dy}{dt}\right) & \frac{\partial}{\partial y}\left(\frac{dy}{dt}\right) \end{pmatrix}\right|_{x_0,y_0}$$

Note that for a system of linear differential equations, the model matrix itself is the Jacobian. The values in that matrix are applicable (and exactly correct) everywhere in model space.

For our system, we can write

$$J = \left.\begin{pmatrix} -1 & -\frac{\alpha n y^{n-1}}{(1+y^n)^2} \\ -\frac{\alpha n x^{n-1}}{(1+x^n)^2} & -1 \end{pmatrix}\right|_{x_0,y_0}$$

We can write a function in Matlab to compute this for us (file switch_j.m):

```
function [ J ] = switch_j( a, n, x, y )
%SWITCH_J Summary of this function goes here
%   Detailed explanation goes here
J= [                      -1,    -a/(1+y^n)^2*y^(n-1)*n;
     -a/(1+x^n)^2*x^(n-1)*n,                         -1];
```

Now we can evaluate the Jacobian and its eigenvalues for the other stationary points…

```
>> x=x2(1); y=x2(2);
>> J = switch_j(alpha,n,x,y);
>> J

J =

   -1.0000   -0.2547
   -1.7454   -1.0000

>> eig(J)

ans =

   -0.3333
   -1.6667
```

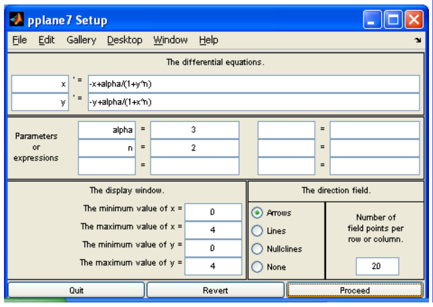
It is apparent that the second (and by symmetry, the third) stationary points are stable.

Phase Portraits

When we investigated our 1D non-linear dynamic system, we constructed a 1D phase portrait to help us understand how the system behaves over a range of values. 2D phase portraits are very useful in understanding the behavior of more complex dynamic systems. There is a freely available add-on tool for Matlab called `pplane` that prepares such diagrams. `pplane` is developed by John C. Polking in the Department of Mathematics at Rice University, and is available via Matlab's File Exchange. Unfortunately, this tool seems to break with each new version of Matlab, and updates to fix this don't come out promptly. As of the time of this writing, there is no version of pplane that work with Matlab 2020; older vesions of pplane (plane7, 8, or 9) should work with older versions of Matlab. To invoke the tool, enter the command, e.g. `pplane7`, at the Matlab prompt.

Alternatively, a Java version of pplane that can be obtained from https://www.cs.unm.edu/~joel/dfield/. To run this you'll need a Java interpreter installed on your computer (I run this with OpenJDK instead of the official Sun JDK).

Below are two screenshots of `pplane` being used to model our switch system. The arrows indicate the direction that a solution moves in. The orange and pink lines are nullclines, and the blue lines are trajectories that the system can take. This tool can also compute Jacobians, and eigensystems at stationary points.



The differential equations.

x ' = -x+alpha/(1+y^n)
y ' = -y+alpha/(1+x^n)

Parameters or expressions

alpha = 3
n = 2

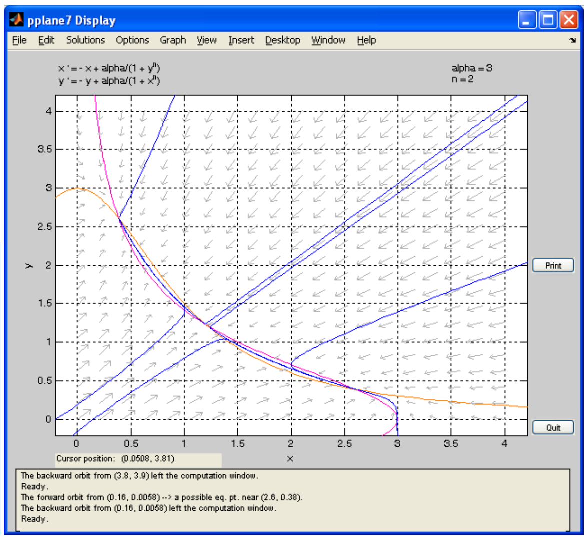
The display window.

The minimum value of x = 0
The maximum value of x = 4
The minimum value of y = 0
The maximum value of y = 4

The direction field.

[✓ Arrows] Lines Nullclines None

Number of field points per row or column. 20


pplane7 Display
File Edit Solutions Options Graph View Insert Desktop Window Help
x ' = - x + alpha/(1 + y^n)
y ' = - y + alpha/(1 + x^n)
alpha = 3
n = 2
y
x
Print
Quit

Cursor position: (0.0508, 3.81)

```
The backward orbit from (3.8, 3.9) left the computation window.
Ready.
The forward orbit from (0.16, 0.0058) --> a possible eq. pt. near (2.6, 0.38).
The backward orbit from (0.16, 0.0058) left the computation window.
Ready.
```