Quantitative Understanding in Biology
Module IV: ODEs
Lecture I: Introduction to ODEs

Ordinary Differential Equations

We began our exploration of dynamic systems with a look at linear difference equations. This is a nice introduction to dynamic systems for a few reasons. First, linear systems are "well-behaved" in that they can be treated analytically. Second, the use of discrete time steps in difference equations allowed us to side-step a fair amount of calculus.

Unfortunately, the real world is not so convenient, so we'll need to gear up to deal with both non-linear systems and with true differential equations. We'll try to keep the necessary calculus to a minimum, but some will be needed. Fortunately, many of the operative concepts are similar, so although the math may be a little more intimidating, you are now in a good position to understand it.

We'll begin by considering one of the simplest, and most essential, dynamic systems, and compare and contrast it to its discrete counterpart. Let's return to an example we looked at previously, that of growth proportional to current amount or size. The discrete form of this model is…

$$x_{n+1} = \lambda \cdot x_n$$

…and the solution is…

$$x_n = c \cdot \lambda^n$$

To express the idea of a rate of growth proportional to current size or amount using differential equations, we would write…

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

Solving this (using calculus) gives us…

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

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

$$\ln x = \lambda \cdot t + C$$

…where C is an arbitrary constant of the integration.

This can be simplified into the conventional form…

$$x = e^{\lambda \cdot t + C}$$

$$x = e^{\lambda \cdot t} e^{C}$$

$$x = c \cdot e^{\lambda \cdot t}$$

In the last step we have redefined the arbitrary constant for our convenience.

To review, we have

Discrete	Continuous
$x_n = c \cdot \lambda^n$	$x = c \cdot e^{\lambda \cdot t}$

Note that in both cases the evolution of our state variable is some value raised to a power that has to do with time. In the discrete case, the exponent is the number of time steps in our model simulation; this is a dimensionless counting number. In the continuous case, the exponent is a "scaled time". It is the time of the simulation, but in 'units of 1/λ'. You will see this motif very often in this form and in one other…

$$x = c \cdot e^{t/\tau}$$

In this latter form, τ is the 'time constant' of the system. It has units of time, just like t, so the ratio is dimensionless. The ratio t/τ is sometimes referred to as the "reduced time".

Note that the values of the λs in the discreet and continuous cases have different interpretations. In the discrete model, a value of λ=1 yields a steady-state solution; the system neither 'explodes' nor 'fizzles'. When λ > 1, the system will grow, and when 0 < λ < 1, the system will decay.

In the continuous time model, the equilibrium condition is reached when the derivative is zero, or when λ=0. Negative values of λ give decaying systems and positive values of λ give growing systems.

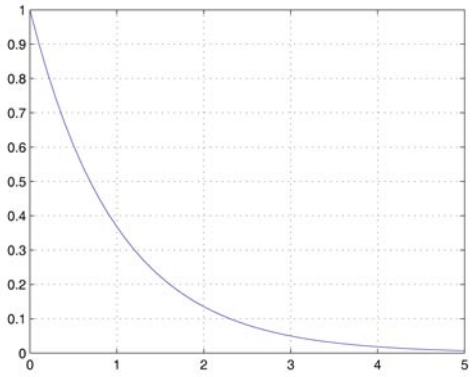
Exponential Decay

The exponential decay is probably the most fundamental motif you'll see when you study differential equations and dynamic systems in general. Many dynamic systems in nature that reach an equilibrium approach it by an exponential decay. To emphasize the nature of the decay we usually write…

$$x = c \cdot e^{-t/\tau}$$

…rather than taking τ to be negative.

It is worth inspecting a plot of x vs. reduced time, and becoming familiar with some of the numerical values involved.



t	x
0	1
0.6931	$\frac{1}{2}$
1.0986	$\frac{1}{3}$
2	0.1353
3	0.0498

Some salient features of the exponential decay are:

- The decay falls to half its initial value in about 0.7 time constants.
- The decay falls to one-third of its initial value in about 1.1 time constants.
- In two time constants, the value has decayed to about 13% of its initial value.
- In three time constants, the value has decayed to about 5% of its initial value.

Note that an exponential decay never "finishes" as it asymptotically approaches zero. However, as it is useful to think about dynamic events, we can make an arbitrary definition about how much of a decay has to have been achieved before an event is 'over'. If we define the end of an event as the time when it reaches 95% recovery to the equilibrium value, we see that events take about three time constants. If you want to use a different cutoff, that is OK. What is important is that the length of an event is on the order of a few time constants. For example, if you define your cutoff as 99.44% recovery, you'll find that events last 5.2 time constants. This line of reasoning leads us to the very important realization that τ tells us "how long stuff takes to happen in our system".

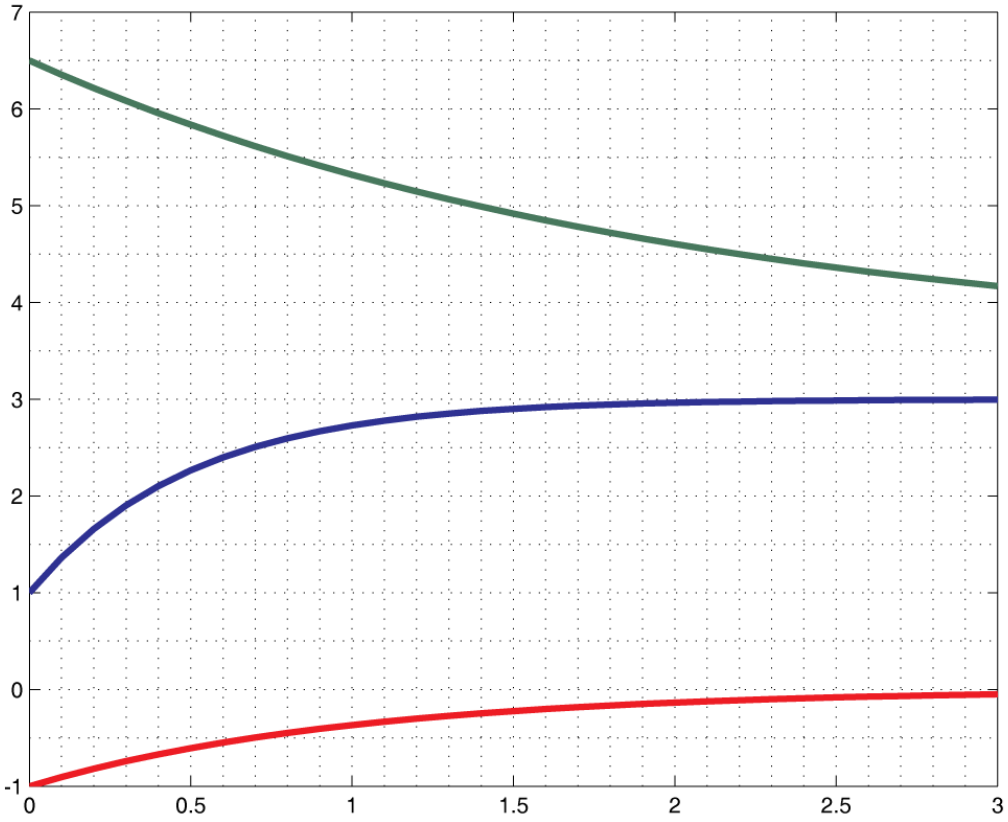
Note that exponential decays don't always decay towards zero, and don't always decay from higher values to lower values. For example, consider the system

$$\frac{dx}{dt} = -\lambda \cdot (x - x_0)$$

The general solution to this system is…

$$x = x_0 + c \cdot e^{-\lambda \cdot t}$$

A few particular solutions to this system for different values of x_0, c, and t are shown in the figure below. From inspection of the plot only, you should be able to estimate τ for the red (bottom) and blue (middle) curves. The green (upper) curve may be a little trickier; how might you reliably determine τ for this case? [Ans: from top to bottom, τ = 2, 0.5, 1].



Formulating Systems of ODEs

The art of modeling dynamic systems usually begins with a description of a system and a transformation of that description into a set of differential equations. If the system considers spatial variability in addition to temporal variability, you will usually end up with a set of partial differential equations. For this section of the class we will focus on systems that do not consider spatial variability, and therefore will result in ordinary differential equations.

Let's consider an example of how a system of ODEs can be formulated from a system description. We'll return to our basic enzyme-substrate system:

$$E + S \overset{k_1, k_{-1}}{\longleftrightarrow} ES \overset{k_2}{\rightarrow} E + P$$

We can write the differential equation of each species by simple application of the law of mass action.

$$\frac{d[E]}{dt} = -k_1[E][S] + k_{-1}[ES] + k_2[ES]$$

$$\frac{d[S]}{dt} = -k_1[E][S] + k_{-1}[ES]$$

$$\frac{d[ES]}{dt} = k_1[E][S] - k_{-1}[ES] - k_2[ES]$$

$$\frac{d[P]}{dt} = k_2[ES]$$

We should note that the above differential equations assume the system is at constant volume. As this is not a chemistry class, we will just accept this.

To simulate the above system, we need the model parameters. In this case, there are three: k_1, k_{-1}, and k_2. We also need initial conditions for the state variables. In this case, four would be needed: $[E]_0$, $[S]_0$, $[ES]_0$, and $[P]_0$.

Running a Simulation: Basic Theory

The most common way of thinking about running a simulation of a system of ODEs is the Euler method. We "discretize" the system by turning the differential equations into the difference equation analogs. The discrete analog of the last equation in our system would be…

$$\frac{\Delta[P]}{\Delta t} = k_2 \cdot [ES]$$

$$[P]_{n+1} - [P]_n = k_2 \cdot [ES]_n \cdot \Delta t$$

$$[P]_{n+1} = [P]_n + k_2 \cdot [ES]_n \cdot \Delta t$$

A similar technique can be applied to the other equations in the system. These difference equations can be solved quite straightforwardly by simple iteration. If we are lucky and the resultant system of equations is linear and can be cast in matrix form, we can bring the techniques we learned in the previous section to bear. Unfortunately, this is not the case in this system. The second equation discretizes as follows:

$$\frac{\Delta[S]}{\Delta t} = -k_1[E][S] + k_{-1}[ES]$$

$$[S]_{n+1} = [S]_n - k_1[E]_n[S]_n \cdot \Delta t + k_{-1}[ES]_n \cdot \Delta t$$

Sadly, the product of the two state variables precludes a linear, matrix formulation of the system.

Once we have discretized our differential equations, we can choose a sufficiently small Δt and use a computer to iterate over time.

This method, elegant and simple as it is, has a pitfall. The devil is in the phrase 'sufficiently small'. Often sufficiently small means really, really small. Sometimes it is impractically small. The biggest impact of this is often computer time; you may need lots of steps to iterate your system. However, too small can get you into trouble with the floating point accuracy of your computer too. Additionally, the simple Euler integration method can introduce systematic errors.

To see the problem, let's consider the example of a simple exponential

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

In this case we know the analytical solution to be (for the case where $x_0 = 0$)

$$x = 1 - e^{-x}$$

This is an exponential approach to x=1. The discrete version of the differential equation is

$$x_{n+1} = x_n(1 - \Delta t) + \Delta t$$

It is easy to show that for every step, no matter how small Δt is, this equation will always over-predict the change in x.

It can be shown that Euler integration has errors that are $O(\Delta t^2)$. Thus, to get to a "reasonable" answer, one may need to choose a very small Δt.

Running a Simulation: Practical Theory

There are many schemes that try to fix the problem. The so-called predictor-corrector methods use an Euler-like method to predict the new value of x. The derivative is then evaluated at the new value, and is used to correct the prediction in some manner.

Probably the most important practical method for integrating systems of ODEs is the fourth order Runge-Kutta method; this is often abbreviated as RK4. It can be shown that the RK4 method has an error of $O(\Delta t^5)$. While there are several RK algorithms of different orders, RK4 is the most commonly used. If someone says they are using the Runge-Kutta algorithm without specifiying the order, you can be sure they are using RK4 ($p<0.0001$). The equations for RK4 are given here for completeness only…

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$
$$k_1 = f(t_n, y_n)$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

Another important enhancement is the idea of variable step size. The idea is that if things are not changing very quickly in your simulation, you can take larger step sizes. This can be a huge time-saver. Implementation of the logic to achieve variable step sizes is beyond the scope of this course.

Running a Simulation: Practical Methods

The good news is that Matlab implements RK4 with variable step size so you don't have to worry about the details. The key to getting Matlab to do the heavy lifting is that you need to write a Matlab function that gives the derivatives of each state variable. This function is stored in a m-file. For example, the following m-file will give the derivative to an exponential approach to a specified value with a specified time constant (File: test1_func.m):

```
function [ dydt ] = test1_func(t, y, y0, tau)
%TEST1_FUNC Simple exponential decay for testing RK4 ODE solver
%   Provides derivatives for an exponential decay to y0 with time constant tau.
dydt = [ -(y - y0)/tau ];
```

Then, in Matlab, with this on the path, we write…

```
start = 0;
y0 = 5;
tau = 1;
tspan = [0 10];
[t y] = ode45(@test1_func, tspan, start, [], y0, tau);
plot(t,y)
```

It is also instructive to inspect the output arrays t and y. You'll see that the time step is not uniform.

Warning: plot(y) will give you erroneous results!

A more complex example – our Michaelis-Menten model (file: mm.m):

```
function [ dydt ] = mm( t, y, k1, k_1, k2 )
%MM derivatives for Michaelis Menten kinetics
%   MM kinetics:  E + S <----k1,k_1----> ES --k2--> E + P
%   state vars are: y(1) = S, y(2) = E, y(3) = ES, y(4) = P

% Fetch variables
S = y(1);
E = y(2);
ES = y(3);
P = y(4);

dydt = [ -k1 * S * E + k_1 * ES;
         -k1 * S * E + k_1 * ES + k2 * ES;
          k1 * S * E - k_1 * ES - k2 * ES;
          k2 * ES
          ];
```

And then in Matlab:

```
start = [1, 0.1, 0, 0];
k1 = 0.1;
k_1 = 0.1;
k2 = 0.3;
tspan = [0 1000];
 [t, y] = ode45(@mm, tspan, start, [], k1, k_1, k2);
plot(t,y);
```

Observe how quickly the complex forms and reaches a “quasi-equilibrium” concentration. Then observe that the product forms with a much slower time constant. Imagine how many simulation steps you would need if the whole simulation were run with a time step necessary to get the first bit of the simulation correct.

