The Predator-Prey Equation
A system of ODE's needing an accurate solver

MATH1902: Numerical Solution of Differential Equations

http://people.sc.fsu.edu/~jburkardt/classes/math1902_2020/predator/predator.pdf

---

More predators means less prey means less predators means more prey...

The predator prey system

A coupled system of differential equations explains changes in several variables by their interactions with each other. Here we consider the population changes in two communities, predators (foxes) and prey (rabbits). Mathematically, these equations define a cycle that repeats endlessly. If a numerical computation is not very accurate, the cycle quickly deteriorates into a death spiral.

1 The predator-prey ODE system

Suppose that, at any time t, we have populations of $u(t)$ prey and $v(t)$ predators, which might be rabbits and foxes. We assume that changes in these two populations come both from their separate natures (birth, natural death), but also from their interaction with each other (fox catches rabbit and eats it).

This drama is hidden by the following simple pair of differential equations:

$$\frac{du}{dt} = \alpha u - \beta uv$$
$$\frac{dv}{dt} = -\gamma v + \delta uv$$

where $\alpha, \beta, \gamma, \delta$ are positive parameters which control the relationship between the predators and prey.

- α is the rate at a single prey reproduces, so $\alpha * u$ is the group rate;
- β is the rate at which a single predator is likely to catch a single prey, so $\beta * u * v$ is the rate at which any predator catches any prey;

- γ is the rate at which a single predator dies of old age, so $\gamma * v$ is the group rate;
- δ is the rate at which catching a single prey increases the reproductive rate of a single predator, so $\delta * u * v$ is the group rate;

For our experiments, we will use the values $\alpha = 2, \beta = 1, \gamma = 1, \delta = 1$
This gives us a pair of differential equations. To pose a problem, we specify initial conditions:

$$t_0 = 0.0$$
$$u(t_0) = u_0 = 4.0$$
$$v(t_0) = v_0 = 2.0$$

and we suppose we want to compute the solution out to a final time $tstop = 250$. Recall that for our ODE solver, we will store the starting and stopping times as the two elements of the array `tspan`.

2 Our data `y0`, `y`, and `dydt` are now row and column vectors!

Before, our data was a scalar, a single value. Now our data will be stored in vectors, as pairs of numbers at different times. In particular, we need to modify how we deal with the initial condition `y0` and later values `y`, and the value of the derivative `dydt`.

MATLAB has many built-in ODE solvers, and they have already agreed on a convention for how to handle this. For values of the solution, whether at the initial time or later, we will assume that the information is stored as a row vector of length 2, so mathematically we write

$$y0 = \begin{pmatrix} u0, & v0 \end{pmatrix}$$

while in MATLAB we would write

```
y0 = [ u0, v0 ];  % The comma indicates this is a row vector.
```

Meanwhile, the derivative function will return a vector of derivatives, and these must be a column vector, mathematically

$$y' = \begin{pmatrix} \frac{du}{dt} \\ \frac{dv}{dt} \end{pmatrix}$$

while in MATLAB we would write

```
dydt = [ dudt; dydt ];  % The semicolon indicates this is a column vector.
```

where `dudt` and `dvdt` are the values of the right hand sides of the two equations.

Understanding these conventions, we can go on to write out our derivative function.

3 *predator_deriv.m* evaluates the predator/prey right hand side

Just as we did for a single ODE, we need to write a derivative function that evaluates the right hand side. Now our input `y` and output `dydt` are vectors. We will find it convenient to copy and name the first and second components of the `y` vector as `u` and `v`, work out the two derivatives, and then pack them into the column vector `dydt` at the end.

```
function dydt = predator_deriv ( t, y )

  alpha = 2.0;
  beta = 1.0;
```

```
  gamma = 1.0;
  delta = 1.0;

  u = y(1);  %  These commands copy out the two values in y;
  v = y(2);

  dudt = alpha * u - beta  * u * v;
  dvdt = gamma * v + delta * u * v;

  dydt = [ dudt; dvdt ]; %using a semicolon makes dydt a column vector.

  return
end
```

Listing 1: predator_deriv evaluates the predator ODE right hand side.

Once we have our derivative function, it's time to look at the ODE solver.

4 *euler.m* must handle ODE systems now

We have used the Euler code before with a single ODE. In order to work with a system of m equations, we need to modify the old code.
Logically, our ODE solver works the same way, but now we have to deal with arrays, including getting the size of an array, setting aside space for a new array, copying one row of an array into another array, and transposing an array (which is done using an apostrophe!)

Here is a version of the euler solver which can handle systems of ODE's:

```
function [ t, y ] = euler ( dydt, tspan, y0, n )

  m = length ( y0 );
  t = zeros ( n+1, 1 );
  y = zeros ( n+1, m );

  dt = ( tspan(2) - tspan(1) ) / n;
  t(1) = tspan(1);
  y(1,:) = y0;

  for i = 1 : n
    t(i+1) = t(i) + dt;
    y(i+1,:) = y(i,:) + dt * ( dydt ( t(i), y(i,:) ) )';  % use apostrophe for transpose
  end

  return
end
```

Listing 2: euler.m: Euler code modifed to handle systems of ODEs.

5 *predator_euler.m* applies Euler method to predator ODE

Now that we have a derivative function and an ODE solver, we write one last MATLAB function to set up the problem, solve it, and plot it. We will let the number of steps `n` be an input parameter. We will want to compute the solution over a long time interval, $0 \leq t \leq 50$, so we might initially try using 100 steps.

```
function predator_euler ( n )

  dydt = @predator_deriv;
  tspan = [ 0.0, 50.0 ];
  y0 = [ 4.0, 2.0 ];
```

```
6
7     [ t, y ] = euler ( dydt, tspan, y0, n );
8
9     plot ( t, y, 'linewidth', 3 );
10
11    return
12  end
```

Listing 3: predator_euler.m solves and plots.

The results should look terrible! As it turns out, the solution is "sensitive", and the errors in our approximation process keep jumping to larger and larger solution values. To try to control these, we can take smaller steps. Retry the calculation with `n=1000`. If the plot still doesn't make sense, try 10,000 steps. If necessary, go as far as a million steps, and see whether the approximate solution settles down.

6 *predator_euler_phase.m* displays the phase plane solution

The final plot from your previous experiment should suggest that the rabbit and fox populations go through a cycle, that they are periodic. If we have two variables $u(t)$ and $v(t)$ that are behaving periodically, then it can be interesting to make a *phase plane plot*, in which the x axis is u and the y axis is v. If periodic behavior is really going on, then the plot should approximately form a closed curve. To investigate this, we make a new code that has only one important change, in the call to `plot()`:

```
1   function predator_euler_phase ( n )
2
3     dydt = @predator_deriv;
4     tspan = [ 0.0, 50.0 ];
5     y0 = [ 4.0; 2.0 ];
6
7     [ t, y ] = euler ( dydt, tspan, y0, n );
8
9     plot ( y(:,1), y(:,2), 'linewidth', 3 );
10
11    return
12  end
```

Listing 4: predator_euler_phase.m solves and plots phase plane.

Using the same value of n that worked for your previous experiment, what do you see in your phase plane plot?

7 The Predator ODE has a conserved quantity

If we don't know the exact formula for the solution of the predator-prey system, are there other ways to judge the error, and to evaluate how well an ODE solver is doing? Although an ODE describes changes over time, many systems have certain properties that don't change at all over time. For example, it is possible to create a model of the downhill motion of ice in a glacier. For the moment, we'll assume we're studying the glacier in a "quiet season", when no snow is falling to add ice, and no ice is melting at the bottom. Then, although the individual particles of ice will move over time, the total mass of the ice never changes. An ODE solver that models a glacier must produce results that at least approximately conserve the ice mass, or no one will believe the data.

In the predator-prey ODE, we don't conserve the number of rabbits, or the number of foxes, or the sum of the rabbits and foxes. However, it turns out that there is a hidden quantity that will be conserved, depending

on the values of the parameters $\alpha, \beta, \gamma, \delta$. Here is how to discover that quantity:

$$\begin{aligned}
\frac{du}{dt} &= \alpha u - \beta uv && \text{Differential equation 1} \\
\frac{dv}{dt} &= -\gamma v + \delta uv && \text{Differential equation 2} \\
\frac{du}{dv} &= \frac{u}{v}\left(\frac{\alpha - \beta v}{-\gamma + \delta u}\right) && \text{Formal ratio equation1/equation2} \\
\frac{-\gamma + \delta u}{u} du &= \frac{\alpha - \beta v}{v} dv && \text{Separation of variables} \\
\int \frac{-\gamma + \delta u}{u} du &= \int \frac{\alpha - \beta v}{v} dv && \text{Indefinite integrals} \\
-\gamma \ln(u) + \delta u &= \alpha \ln(v) - \beta v + C && \text{Integration with constant} \\
-\gamma \ln(u) + \delta u - \alpha \ln(v) + \beta v &= C && \text{Conserved quantity!}
\end{aligned}$$

This tells us that, for any such system, the sequence of points $(u(t), v(t))$ must maintain a constant value of the conserved quantity $C(t) = C(u(t), v(t)) = -\gamma \ln(u(t)) + \delta u(t) - \alpha \ln(v(t)) + \beta v(t)$. IWe can define $C_0 = C(t_0)$, the value of the conservation quantity at the initial time. Then if we do a numerical computation, at any time t, we can compare $C(t)$ to C_0 to see how well our approximation is conserving this quantity.

For our initial condition $(u_0, v_0) = (4, 2)$, we have

$$\begin{aligned}
C_0 &= -\gamma \ln(u_0) + \delta u_0 - \alpha \ln(v_0) + \beta v_0 \\
&= -\ln(4) + 4 - 2\ln(2) + 2 \\
&\approx 3.22741
\end{aligned}$$

Now let's write a program that solves our problem again, and produces a plot of $C(t)$..

8 *predator_euler_conservation.m* displays $C(t)$

```
function predator_euler_conservation ( n )

  dydt = @predator_deriv;
  tspan = [ 0.0, 50.0 ];
  u0 = 4.0;
  v0 - 2.0;
  y0 = [ u0, v0 ];

  [ t, y ] = euler ( dydt, tspan, y0, n );

  u = y(:,1);
  v = y(:,2);
  c0 = delta * u0 - gamma * log ( u0 ) + beta * v0 - alpha * log ( v0 );
  c =  delta * u  - gamma * log ( u  ) + beta * v  - alpha * log ( v  );

  plot ( t, c, 'r-', ...                                  %  Plot c(t)
         [t(1),t(end)], [c0,c0], 'b--', ...               %  Plot horizontal y=c0 line
         [t(1),t(end)], [0,0], 'k-', 'linewidth', 3 ); %  Plot horizontal y=0 line

  return
end
```

Listing 5: predator_euler_conservation.m plots C(t).

Now run this program for various values of `n`, starting with 100. If the plot suggests that $c(t)$ is way off, than try a value of `n` that is 10 times larger. It shouldn't take too many tries before the conservation plot looks reasonable, though not perfect!

9 Homework #3

Save a copy of your conservation plot from the previous exercise as a PNG file. As a verification, send this file as an attachment to `trenchea@pitt.edu`.