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

MATH1090: Directed Study in Differential Equations

http://people.sc.fsu.edu/~jburkardt/classes/math1090_2020/predator/predator.pdf

---

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

The predator prey system

A pair of differential equations model the population changes in predator and prey communities. 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 system

Suppose that, at any time t, we have populations of $u(t)$ prey and $v(t)$ predators, which might be rabbits and foxes.

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

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

- α is the rate at which the prey reproduce;
- β is the rate at which a single predator is likely to catch a single prey;
- γ is the rate at which the predators die off;
- δ is the rate at which catching a single prey increases the predator population;

For our experiments, we will use the values $\alpha = 2, \beta = 1, \gamma = 1, \delta = 1$
This gives us a pair of differential equations. Now we include an initial time $t_0 = 0$, and initial conditions

$$u(t_0) = 4.0$$
$$v(t_0) = 2.0$$

and we suppose we want to compute the solution out to time $t = 250$.

2 Modifying the Euler code

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 it slightly. We will assume that the initial condition `y0` is input as an m-column vector. The solution values, which we called `y`, will now be a matrix of dimension $m \times n + 1$. The derivative function will now return a vector of derivatives, and these must also be a column vector.
The modified code is available on the web page.

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

  m = size ( y0, 1 );

  tfirst = tspan(1);
  tlast = tspan(2);
  dt = ( tlast - tfirst ) / n;
  t = zeros ( 1, n + 1 );
  y = zeros ( m, n + 1 );
  t(1) = tspan(1);
  y(:,1) = y0(:,1);

  for i = 1 : n
    t(1,i+1) = t(i) + dt;
    y(:,i+1) = y(:,i) + dt * dydt ( t(i), y(:,i) );
  end

  return
end
```

Listing 1: euler.m: Euler code modifed to handle systems of ODEs.

3 Solution with the Euler code

As with a single equation, we write a MATLAB function to evaluate the right hand side:

```
function duvdt = predator_deriv ( t, uv )
  alpha = 2.0;
  beta = 1.0;
  gamma = 1.0;
  delta = 1.0;
  u = uv(1);
  v = uv(2);
  dudt = alpha * u - beta * u * v;
  dvdt = gamma * v + delta * u * v;
  duvdt = [ dudt; dvdt ];
  return
end
```

Listing 2: predator_deriv evaluates the predator ODE right hand side.

Now we can request an approximate solution:

```
duvdt = @predator_deriv;
tspan = [ 0.0, 250.0 ];
uv0 = [ 4.0; 2.0 ];
n = 100;
[ t, uv ] = euler ( duvdt, tspan, uv0, n );

figure ( 1 )
plot ( t, uv, 'linewidth', 3 );
grid ( 'on' );
xlabel ( '<-- T -->' );
ylabel ( '<--U(T), V(T) -->' );
title ( 'Predator Prey plot (Euler).' );
print ( '-dpng', 'predator_plot.png' );

figure ( 2 )
plot ( uv(1,:), uv(2,:), 'linewidth', 3 );
grid ( 'on' );
xlabel ( '<-- U(T) -->' );
ylabel ( '<-- V(T) -->' );
title ( 'Predator Prey phase plane (Euler).' );
print ( '-dpng', 'predator_phase.png' );
```

Listing 3: predator_phase_euler.m makes plot and phase plane.

The results look terrible! Concentrate on the phase plane plot, which we should expect would show some sort of closed curve. Instead, it is nonsense. We assume that the problem is that we are simulating the solution over a long time, and only taking 100 steps. Try again, using 1000 steps. If that doesn't seem better, try 10,000 steps. Keep trying until the phase plane plot seems to make some sense.

4 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?

It turns out that we can make an interesting analysis of a predator prey system with parameters $\alpha, \beta, \gamma, \delta$:

$$\frac{du}{dt} = \alpha u - \beta uv \quad \text{Differential equation 1}$$

$$\frac{dv}{dt} = -\gamma v + \delta uv \quad \text{Differential equation 2}$$

$$\frac{du}{dv} = \frac{u}{v}\left(\frac{\alpha - \beta v}{-\gamma + \delta u}\right) \quad \text{Formal ratio equation1/equation2}$$

$$\frac{-\gamma + \delta u}{u} du = \frac{\alpha - \beta v}{v} dv \quad \text{Separation of variables}$$

$$\int \frac{-\gamma + \delta u}{u} du = \int \frac{\alpha - \beta v}{v} dv \quad \text{Indefinite integrals}$$

$$-\gamma \ln(u) + \delta u = \alpha \ln(v) - \beta v + C \quad \text{Integration with constant}$$

$$-\gamma \ln(u) + \delta u - \alpha \ln(v) + \beta v = C \quad \text{Conserved quantity!}$$

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 $V(u, v) = -\gamma \ln(u) + \delta u - \alpha \ln(v) + \beta v$. For a given initial condition (u_0, v_0), we can determine this constant value: $C = V(u_0, v_0)$. Then if we do a numerical computation, at any time t, we can compare C to $V(u(t), v(t)$ to see how well our approximation is conserving this quantity.

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

$$\begin{aligned} C &= -\gamma \ln(u) + \delta u - \alpha \ln(v) + \beta v \\ &= -\ln(4) + 4 - 2\ln(2) + 2 \\ &\approx 3.22741 \end{aligned}$$

For our 100 step Euler solution, we print the value of V(u(t),v(t)) at 11 equally spaced times between 0 and 250.

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

inc = n / 10;
for i = 1 : inc : n + 1
  fprintf ( 1, '%4d  %14.6g  %14.6g  %14.6g\n', i, t(i), c(i), abs ( c(i) - c0 ) );
end
```

Listing 4: Printing the conserved quantity.

When we try to solve the predator prey problem with the Euler code, we have a perfect start, but thereafter our V values are printed as `NaN`, meaning the results have become meaningless, because the solution has blown up. We expected that there would be some error in our calculation, but for this problem, our error is out of control!

```
predator_euler:

    i              t(i)          V(u,v)        |c-V(u,v)|

    1               0          3.22741               0
   11              25             NaN             NaN
   21              50             NaN             NaN
  ...             ...             ...             ...
  101             250             NaN             NaN
```

Listing 5: Disastrous output from a 100 step Euler calculation

5 Higher order methods

Apparently, the predator-prey system is harder to solve accurately, compare to the hump problem we looked at before. Using the Euler method, we have to take a lot of steps to get a solution we can trust. But the Euler method is only "first-order accurate". Perhaps we can use fewer steps if we can find an ODE solver that does a better job of predicting where the ODE is going to go next.

The Runge-Kutta family of methods provides a sequence of ODE solvers of increasing accuracy. A common choice is the fourth-order method, known as *RK4*. We can use this algorithm as an exact substitute for Euler, using a fixed stepsize, and the same set of input and output data.

Assuming that `tspan(1)` is the initial time, `y0` the initial condition, `dt` the stepsize, and `n` the number of steps we want to take, the RK4 calculation, as carried out in *rk4.m*, might look like this:

```
  t(1) = tspan(1);
  y(:,1) = y0(:,1);

  for i = 1 : n

    f1 = dydt ( t(i),            y(:,i) );
```

```
7       f2 = dydt ( t(i) + dt / 2.0, y(:,i) + dt * f1 / 2.0 );
8       f3 = dydt ( t(i) + dt / 2.0, y(:,i) + dt * f2 / 2.0 );
9       f4 = dydt ( t(i) + dt,       y(:,i) + dt * f3 );
10
11      t(1,i+1) = t(i) + dt;
12      y(:,i+1) = y(:,i) + dt * ( f1 + 2.0 * f2 + 2.0 * f3 + f4 ) / 6.0;
13
14    end
```

Listing 6: rk4 ODE algorithm.

The RK4 method is fourth-order accurate, which means that it is much better at predicting the path of the ODE. We can hope that this mean we can get a solution of the predator prey problem using much fewer steps.

Rewrite your Euler solver for predator prey so that now you call rk4 instead. Start again with $n = 100$ steps, and increase n by a factor of 10 each time, until you get a nice phase plane plot. Consider the difference between this value of n, and the value you needed when using the Euler method.

6 Report

We now have two ways to solve the predator-prey problem. Let's focus on the quantity V, which for a perfect solution would be exactly the same for every time step. For both Euler and RK4, we are going to try to compute a solution so that at the last step, the value of V has not changed by more than 1.0 from its initial value.

Run the Euler code with $n = 100$, then $n = 1,000$ and so on. Look at the first and last values of V, and stop when your value of n is large enough that the final value of V is no more than 1 unit away from the initial value. Report the value of n, and the initial and final values of V.

Now repeat this experiment with the RK4 code, and again report the value of n, and the initial and final values of V.

Bring your output and plots, and your revised LaTeX file, to our next meeting, on 2:00pm, Thursday, 06 February, in room Thackeray 624.