10.4 Lotka-Voltera

The Lotka-Voltera model describes the interactions between two species in an ecosystem, a predator and its prey. A common example is rabbits and foxes.

The model is governed by the following system of differential equations:

$$R_t = aR - bRF$$
$$F_t = ebRF - cF$$

where

- R is the population of rabbits,
- F is the population of foxes,
- a is the natural growth rate of rabbits in the absence of predation,
- c is the natural death rate of foxes in the absence of prey,
- b is the death rate of rabbits per interaction with a fox,
- e is the efficiency of turning eaten rabbits into foxes.

At first glance you might think you could solve these equations by calling `ode45` once to solve for R as a function of time and once to solve for F. The problem is that each equation involves both variables, which is what makes this a **system of equations** and not just a list of unrelated equations. To solve a system, you have to solve the equations simultaneously.

Fortunately, `ode45` can handle systems of equations. The difference is that the initial condition is a vector that contains initial values $R(0)$ and $F(0)$, and the output is a matrix that contains one column for R and one for F.

And here's what the rate function looks like with the parameters $a = 0.1$, $b = 0.01$, $c = 0.1$ and $e = 0.2$:

```
function res = lotka(t, V)
    % unpack the elements of V
    r = V(1);
    f = V(2);

    % set the parameters
    a = 0.1;
    b = 0.01;
    c = 0.1;
    e = 0.2;
```

```
    % compute the derivatives
    drdt = a*r - b*r*f;
    dfdt = e*b*r*f - c*f;

    % pack the derivatives into a vector
    res = [drdt; dfdt];
end
```

As usual, the first input variable is time. The second input variable is a vector with two elements, $R(t)$ and $F(t)$. I gave it a capital letter to remind me that it is a vector. The body of the function includes four **paragraphs**, each explained by a comment.

The first paragraph **unpacks** the vector by copying the elements into scalar variables. This isn't necessary, but giving names to these values helps me remember what's what. It also makes the third paragraph, where we compute the derivatives, resemble the mathematical equations we were given, which helps prevent errors.

The second paragraph sets the parameters that describe the reproductive rates of rabbits and foxes, and the characteristics of their interactions. If we were studying a real system, these values would come from observations of real animals, but for this example I chose values that yield interesting results.

The last paragraph **packs** the computed derivatives back into a vector. When `ode45` calls this function, it provides a vector as input and expects to get a vector as output.

Sharp-eyed readers will notice something different about this line:

```
    res = [drdt; dfdt];
```

The semi-colon between the elements of the vector is not an error. It is necessary in this case because `ode45` requires the result of this function to be a column vector.

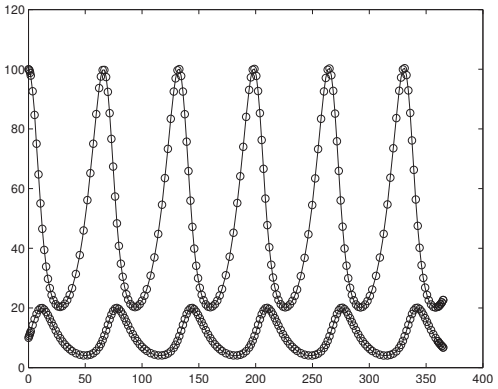
Now we can run `ode45` like this:

```
ode45(@lotka, [0, 365], [100, 10])
```

As always, the first argument is a function handle, the second is the time interval, and the third is the initial condition. The initial condition is a vector: the first element is the number of rabbits at $t = 0$, the second element is the number of foxes.

The order of these elements (rabbits and foxes) is up to you, but you have to be consistent. That is, the initial conditions you provide when you call `ode45` have to be the same as the order, inside `lotka`, where you unpack the input vector and repack the output vector. MATLAB doesn't know what these values mean; it is up to you as the programmer to keep track.

But if you get the order right, you should see something like this:



The x-axis is time in days; the y-axis is population. The top curve shows the population of rabbits; the bottom curve shows foxes. This result is one of several patterns this system can fall into, depending on the starting conditions and the parameters. As an exercise, try experimenting with different values.

10.5 What can go wrong?

The output vector from the rate function has to be a column vector; otherwise you get

```
??? Error using ==> funfun/private/odearguments
LOTKA must return a column vector.

Error in ==> ode45 at 173
[neq, tspan, ntspan, next, t0, tfinal, tdir, y0, f0, odeArgs,
odeFcn, ...
```

Which is pretty good as error messages go. It's not clear *why* it needs to be a column vector, but that's not our problem.

Another possible error is reversing the order of the elements in the initial conditions, or the vectors inside `lotka`. Again, MATLAB doesn't know what the elements are supposed to mean, so it can't catch errors like this; it will just produce incorrect results.

10.6 Output matrices

As we saw before, if you call `ode45` without assigning the results to variables, it plots the results. If you assign the results to variables, it suppresses the figure. Here's what that looks like:

```
>> [T, M] = ode45(@lotka, [0, 365], [100, 10]);
```

You can think of the left side of this assignment as a vector of variables.

As in previous examples, `T` is a vector of time values where `ode45` made estimates. But unlike previous examples, the second output variable is a matrix containing one column for each variable (in this case, R and F) and one row for each time value.

```
>> size(M)

ans = 185     2
```

This structure—one column per variable—is a common way to use matrices. `plot` understands this structure, so if you do this:

```
>> plot(T, M)
```

MATLAB understands that it should plot each column from `M` versus `T`.

You can copy the columns of `M` into other variables like this:

```
>> R = M(:, 1);
>> F = M(:, 2);
```

In this context, the colon represents the range from 1 to `end`, so `M(:, 1)` means "all the rows, column 1" and `M(:, 2)` means "all the rows, column 2."

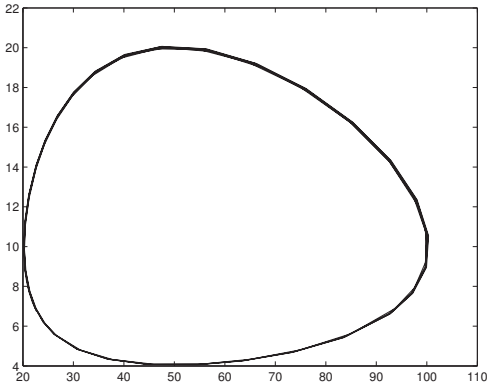
```
>> size(R)

ans = 185     1

>> size(F)

ans = 185     1
```

So `R` and `F` are column vectors.

If you plot these vectors against each other, like this

```
>> plot(R, F)
```

You get a **phase plot** that looks like this:



Each point on this plot represents a certain number of rabbits (on the x axis) and a certain number of foxes (on the y axis).

Since these are the only two variables in the system, each point in this plane describes the complete **state** of the system.

Over time, the state moves around the plane; this figure shows the path traced by the state during the time interval. This path is called a **trajectory**.

Since the behavior of this system is periodic, the resulting trajectory is a loop.

If there are 3 variables in the system, we need 3 dimensions to show the state of the system, so the trajectory is a 3-D curve. You can use `plot3` to trace trajectories in 3 dimensions, but for 4 or more variables, you are on your own.

10.7 Glossary

row vector: In MATLAB, a matrix that has only one row.

column vector: A matrix that has only one column.

transpose: An operation that transforms the rows of a matrix into columns (or the other way around, if you prefer).

system of equations: A set of equations written in terms of a set of variables such that the equations are intertangled.

paragraph: A chunk of code that makes up part of a function, usually with an explanatory comment.

unpack: To copy the elements of a vector into a set of variables.

pack: To copy values from a set of variables into a vector.

state: If a system can be described by a set of variables, the values of those variables are called the state of the system.

phase plot: A plot that shows the state of a system as point in the space of possible states.

trajectory: A path in a phase plot that shows how the state of a system changes over time.

10.8 Exercises

Exercise 10.2 *Based on the examples we have seen so far, you would think that all ODEs describe population as a function of time, but that's not true.*

According to the Wikipedia, "The Lorenz attractor, introduced by Edward Lorenz in 1963, is a non-linear three-dimensional deterministic dynamical system derived from the simplified equations of convection rolls arising in the dynamical equations of the atmosphere. For a certain set of parameters the system exhibits chaotic behavior and displays what is today called a strange attractor..."*

The system is described by this system of differential equations:

$$x_t = \sigma(y - x) \tag{10.1}$$
$$y_t = x(r - z) - y \tag{10.2}$$
$$z_t = xy - bz \tag{10.3}$$

Common values for the parameters are $\sigma = 10$, $b = 8/3$ *and* $r = 28$.

Use `ode45` *to estimate a solution to this system of equations.*

1. *The first step is to write a function named* `lorenz` *that takes* `t` *and* `V` *as input variables, where the components of* `V` *are understood to be the current values of* `x`, `y` *and* `z`. *It should compute the corresponding derivatives and return them in a single column vector.*

2. *The next step is to test your function by calling it from the command line with values like* $t = 0$, $x = 1$, $y = 2$ *and* $z = 3$*? Once you get your function working, you should make it a silent function before calling* `ode45`.

3. *Assuming that Step 2 works, you can use* `ode45` *to estimate the solution for the time interval* $t_0 = 0$, $t_e = 30$ *with the initial condition* $x = 1$, $y = 2$ *and* $z = 3$.

4. *Use* `plot3` *to plot the trajectory of* x, y *and* z.

*`http://en.wikipedia.org/wiki/Lorenz_attractor`