Ch. 48: Finding a solution

Daniel Kaplan

2022-03-10

Student outcomes:

- Recognize a first-order differential equation.
- Given a set of state variables, write down the framework for a system of first-order differential equations.
- Understand that there can be (infinitely) many **solutions** to a differential equation.
- Each solution has its individual **initial condition**.
- Given dynamical functions, use the computer to sketch the flow field.
- Given dynamical functions, find a trajectory numerically and plot it.
- Plot out the individual time series from a numerically computed trajectory.

Overview

Recall that we are using a state-space formalism for writing dynamical systems. In this formalism, there is one or more **state variable**. For each state variable there is a **first-order** differential equation that describes the instantaneous rate of change in that variable as a function of all the state variables.

If the state variables were x, y, and z, the dynamical system would be written as a system of three first-order differential equations.

$\partial_t x = f(x, y, z)$ $\partial_t y = g(x, y, z)$ $\partial_t z = h(x, y, z)$ instantaneous dynamical rate

Chapter 48 is about finding solutions to such sets of differential equations. Three methods are presented:

1. Draw the **flow field**. Mark the selected initial condition and trace a trajectory from that point in the direction indicated by the flow. (This method is appropriate for qualitative understanding of systems of two first-order differential equations.)
2. **Numerically accumulate** the solution, starting at the initial condition and using the **finite-difference approximation to the derivative** to determine successive state values, one step at a time. (This method is appropriate for systems with any number of state variables.)
3. **Symbolic techniques**, which are shown only for a one-state-variable system
    i. Using an exponential ansatz: $Ae^{\omega t}$ which suits a linear differential equation such as $\partial_t x = ax$. There is only one linear, first-order differential equation, so the demonstration at this point is only to familiarize the student with the idea of plugging in an informed guess.
    ii. separation and integration, which suits an equation such as such as $\partial_t x = x(1-x)$. Some instructors may find this a nice niche in which to have students practice symbolic integration, but it will not play a role in the rest of the Block.

Flow field

Insofar as the idea of tracing a trajectory that goes with the flow was introduced in Chapter 47, it may not be necessary to dwell on it.

Some attention should be paid to making graphs of flow fields. This also provides a nice entrée to the Euler method.

Flow by hand

Linear differential equations will be featured in later chapters, so a by-hand activity may be worthwhile.

i. Provide students with a sheet of graph paper, or have them sketch their own. Mark coordinate axes, say $-5 \leq x \leq 5$ and $-5 \leq y \leq 5$.

ii. Give a specific linear system, for example this one that has easy arithmetic and produces a clockwise circular flow.

$$\partial_t x = f(x, y) = y \partial_t y = g(x, y) = -x$$

iii. Ask the students to calculate the x and y components of the flow vector for the specific state $x = 3, y = 2$. It will be $(2, -3)$. a. Suppose the units of x are dollars and the units of y are fish. What will be the units of the components of the flow vector? (Ans: e.g. dollars per day, fish per day.) b. What are the units of the coordinate axes? (Ans: dollars for x and fish for y.) c. Since the coordinate axes are different from the vector components, to draw the vector on the coordinate axes we need to do some conversion. That conversion will be to multiply the vector components by some interval of time, say $dt = 0.5$ days. d. Multiply the (2 dollars/day, -3 fish/day) vector by $dt = 0.5$ days, giving (1 dollar, -1.5 fish). This converted vector is one that can be drawn to scale on the coordinate axes. Do that, with the tail of the vector at coordinate point $x = 3, y = 2$. e. Pick a few more points in the state space and draw the vectors showing the flow at those points.

   1. Are all the flow vectors the same length?
   2. Is there a place where the flow vectors are very short?
   3. Is there a place where the flow vectors are much longer?
   4. Describe the overall pattern.

iv. We used $dt = 0.5$ days as the scaling factor. Is that small enough to give an accurate approximation to the derivative? One way to address that question is to draw the flow vector for the state-space coordinate of the tip of one of your existing flow vectors. If dt is appropriately small, then the original vector and the new vector will have very similar lengths and directions. Ideally, we should use $dt \to 0$ for drawing the vectors, but that leads to vectors that are invisibly short.

Euler by hand

The objective is to draw a trajectory.

i. Mark an initial condition in the dollar/fish system. Don't put it too close to the origin or it will be hard to see what you're doing.
   - Create a table with columns t, x, and y that we'll use to keep track of things numerically. Enter the first row with $t = 0$, and set x and y to be the coordinates of the initial condition.

ii. From that initial condition, calculate the flow vector scaled by $dt = 0.5$ days.
   - Add 4 more columns to the table, $\partial_t x$, $\partial_t y$, and $dt\ \partial_t x$ and $dt\ \partial_t y$.
   - In the first row, fill in the $\partial_t x$ and $\partial_t y$ columns by calculating the value of the dynamical function at x and y.
   - Still in the first row, fill in the $dt\ \partial_t x$ and $dt\ \partial_t y$ columns.

iii. Draw that vector as a line segment, with the tail on the initial condition. The head of the segment is approximately the point that the state would evolve to in half a day. - In your table, start a second row. Set the t column to be dt, the increase in time since the first row. Set x and y in the second row by adding the x,y values from the first row to the $dt\ \partial_t x$ and $dt\ \partial_t y$ values from the first row. The x, y values in the second row will be the position the trajectory reaches at the time indicated in that row.

iv. Go back to steps (ii) and (iii), but the calculation should be done for the end of the line segment you have just drawn rather than for the initial condition. Repeat this over and over until you have a chain of connected line segments.
   - In your table, start the next row. Calculate the value of t by adding dt to the t from the previous row. Similarly calculate x and y for the row by reference to the appropriate values from that previous row.

The table will look like this, with $dt = 0.5$ and the initial condition at $x = 3, y = 2$:

t	x	y	$\partial_t x$	$\partial_t y$	$dt \times \partial_t x$	$dt \times \partial_t y$
0.0	3.00	2.000	2	-3.00	$dt \times 2 = 1.00$	$dt \times -3 = -1.500$
0.5	4	0.500	0.500	-4.00	$0.5\,dt = 0.25$	$-4\ dt = -2.00$
1.0	4.25	-1.500	-1.500	-4.25	$-1.5\ dt = -0.75$	$-4.25\ dt = -2.125$
1.5	3.5	-3.625	$\cdots$	$\cdots$		

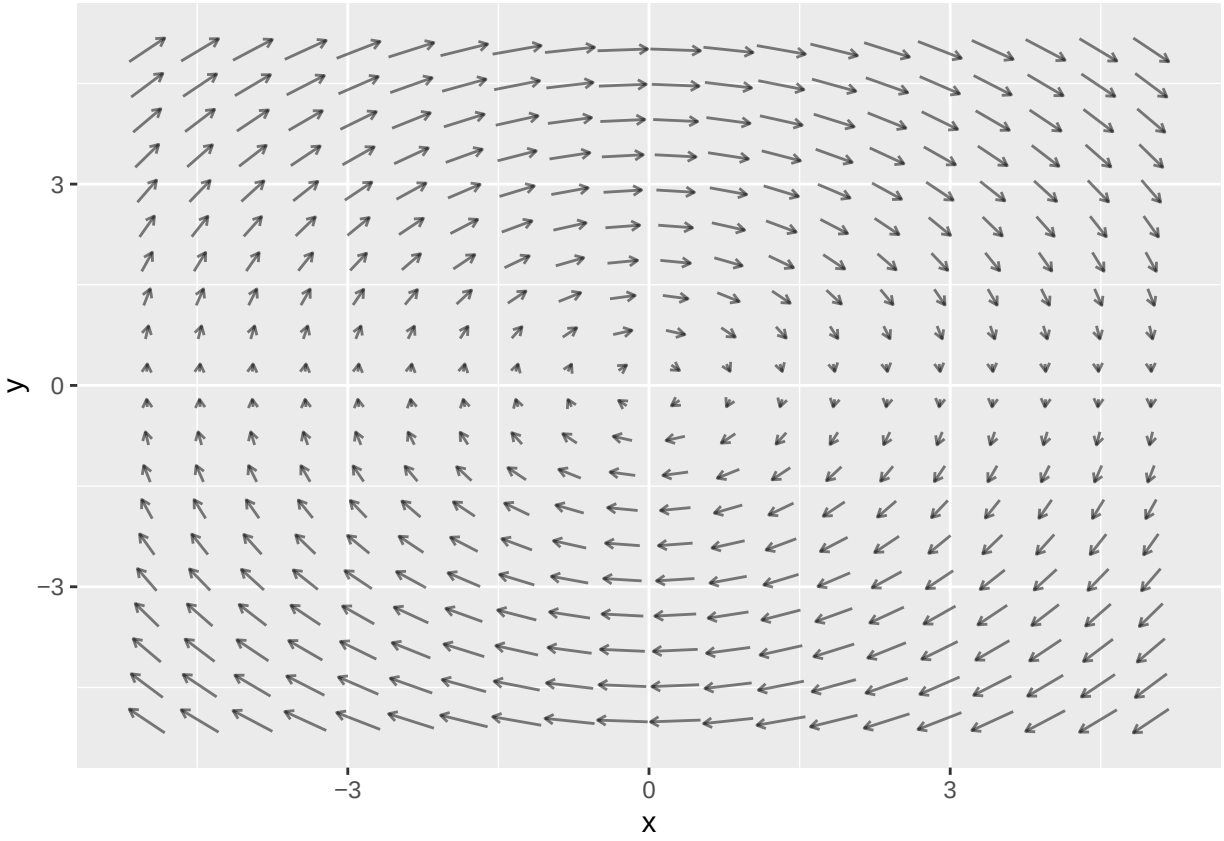
Flow field by software

We'll use two new R/mosaic functions, `vectorfield_plot()` and `streamlines()`. They show the flow in slightly different ways.

Both of them take **two** tilde expressions, one for the x component of the flow, one for the y component of the flow.

`vectorfield_plot()` works in much the same way as our other plotting functions. There are two tilde expressions because there are two functions involved: one describes the horizontal component of the flow vector, the other the vertical component. In using `vectorfield_plot()` you specify with `npts=` how many vectors you want to draw: it's the number found along each axis

```
vectorfield_plot(y ~ x & y, -x ~ x & y, domain(x=-5:5, y=-5:5), npts=20)
```



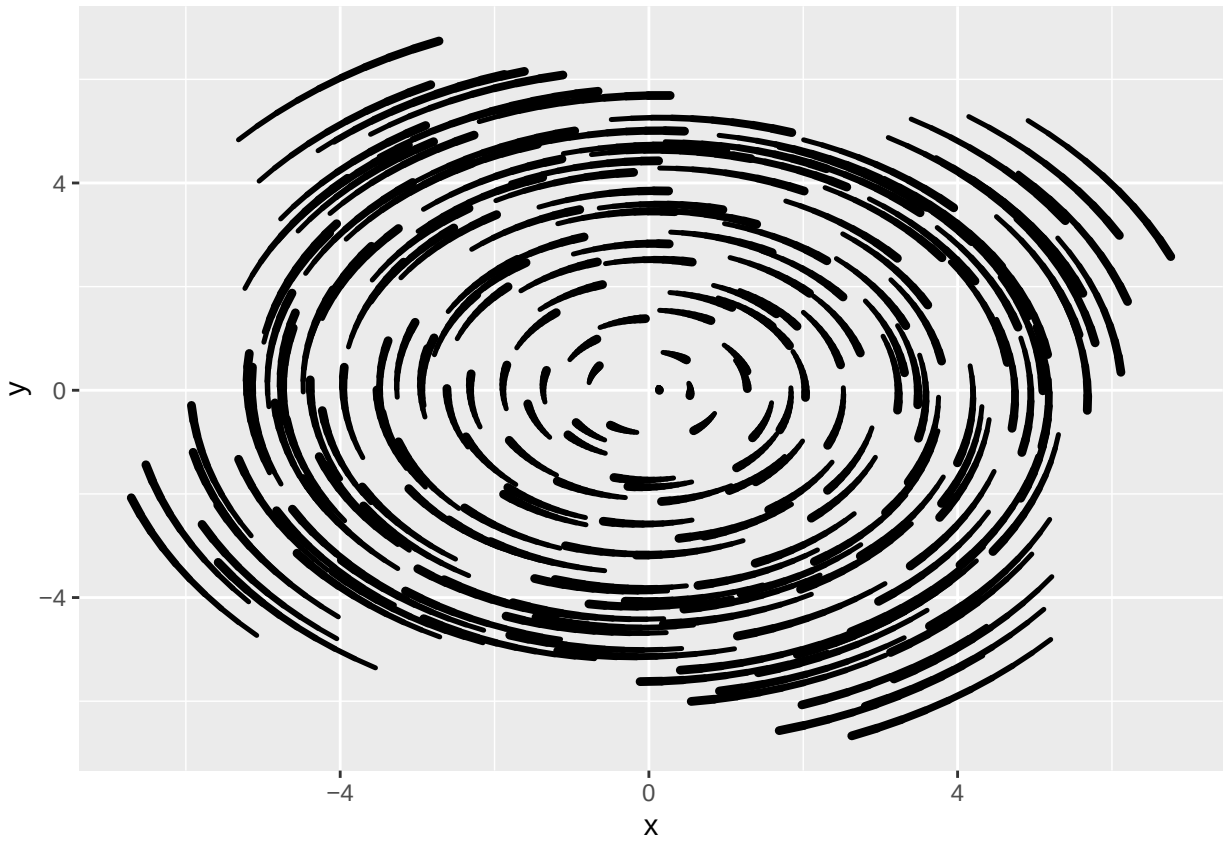
Activity for `vectorfield_plot()`:

i. Have students vary `npts` (but keep it less than 100) and decide for themselves which gives the best overall view of the flow.
ii. Have students make up their own dynamical functions to generate a pretty (to their own taste) flow.
iii. Discuss what the dynamical functions might look like to make a flow field that points to $(0, 0)$.

`streamlines()` provides a similar sort of graphic, but one that includes both dt to set the length of the segments, and which accumulates several successive segments. The segment starting locations are set randomly.

NOTE that the tilde expressions in `streamlines()` have a different format from the ones we've been using. The left side of the expression identifies the component while the right side gives the dynamical function.

```
streamlines(dx ~ y, dy ~ -x, domain=domain(x=-5:5, y=-5:5), dt=0.05, nsteps=10, npts=15)
```

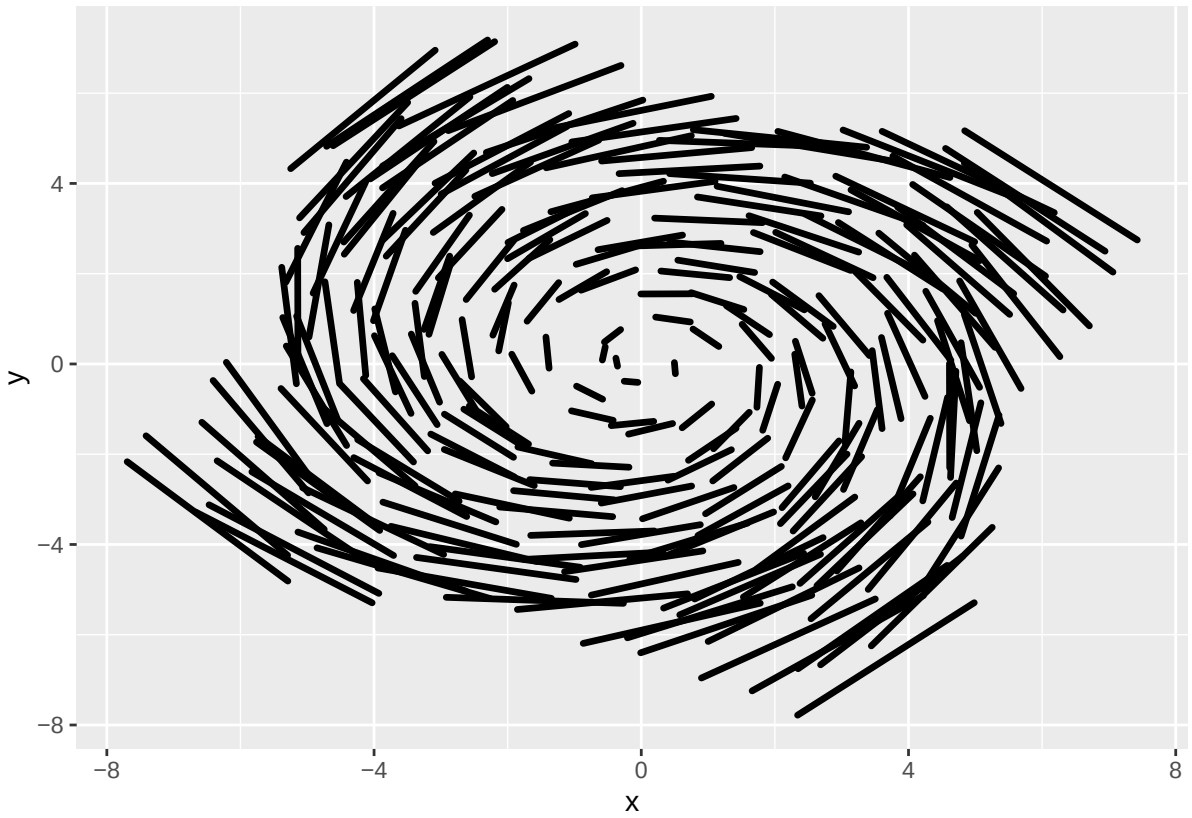


Although dt is set to 0.05, each segment consists of 10 parts, which cover altogether 0.5 time units. That the segments are curved indicates that the value $dt = 0.5$ that we used in the hand-drawing activity gives only a rough approximation to the derivative.

`streamlines()` activity:

i. Change `dt` to 0.25 and `nsteps` to 2 to simulate what we did in the by-hand activity. The resulting straight line segments are about the same length as for `nsteps=10; dt=0.05`, but they don't curve properly. (Due to a bug in the `streamlines()` function, setting `nsteps=2` actually causes just one time step of length `dt` to be drawn.)

```
streamlines(dx ~ y, dy ~ -x, domain=domain(x=-5:5, y=-5:5), dt=0.5, nsteps=2, npts=15)
```



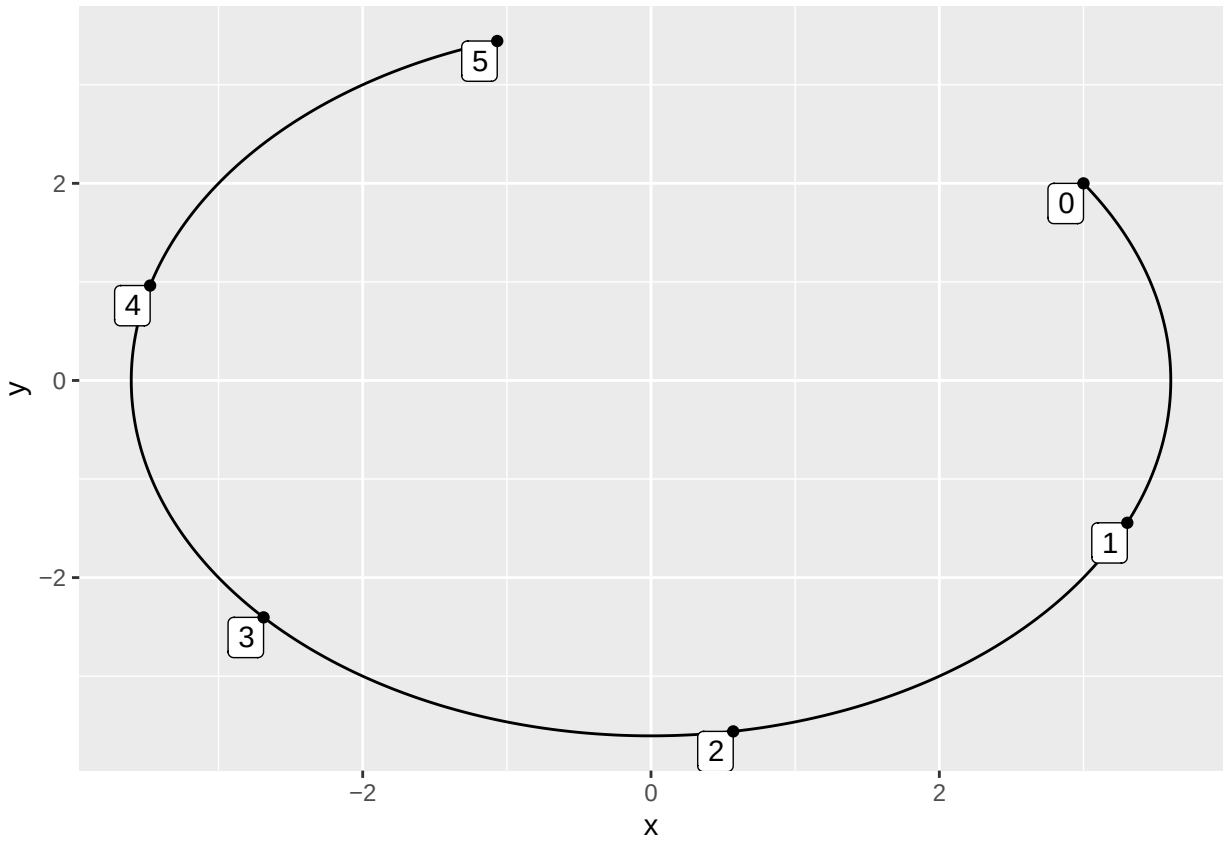
ii. Draw the streamlines for your favorite dynamical functions from the previous activity. Use a suitable small `dt` and set `nsteps` to get a picture that you think tells the story of your dynamical system.

Trajectories by software

We're going to use two new R/mosaic functions, `integrateODE()` and `traj_plot()` to calculate individual trajectories. Note that `streamlines()` is already calculating trajectories and plotting them to give an idea of the shape of the flow. `integrateODE()` is appropriate when you want to follow a single trajectory for a long time.

`integrateODE()` works much like `streamlines()`, but instead of specifying the graphics domain you set the initial condition. The duration of the trajectory is set by `tdur`. (See the documentation for `integrateODE()` to see a more refined way of specifying *dt* and the time domain.)

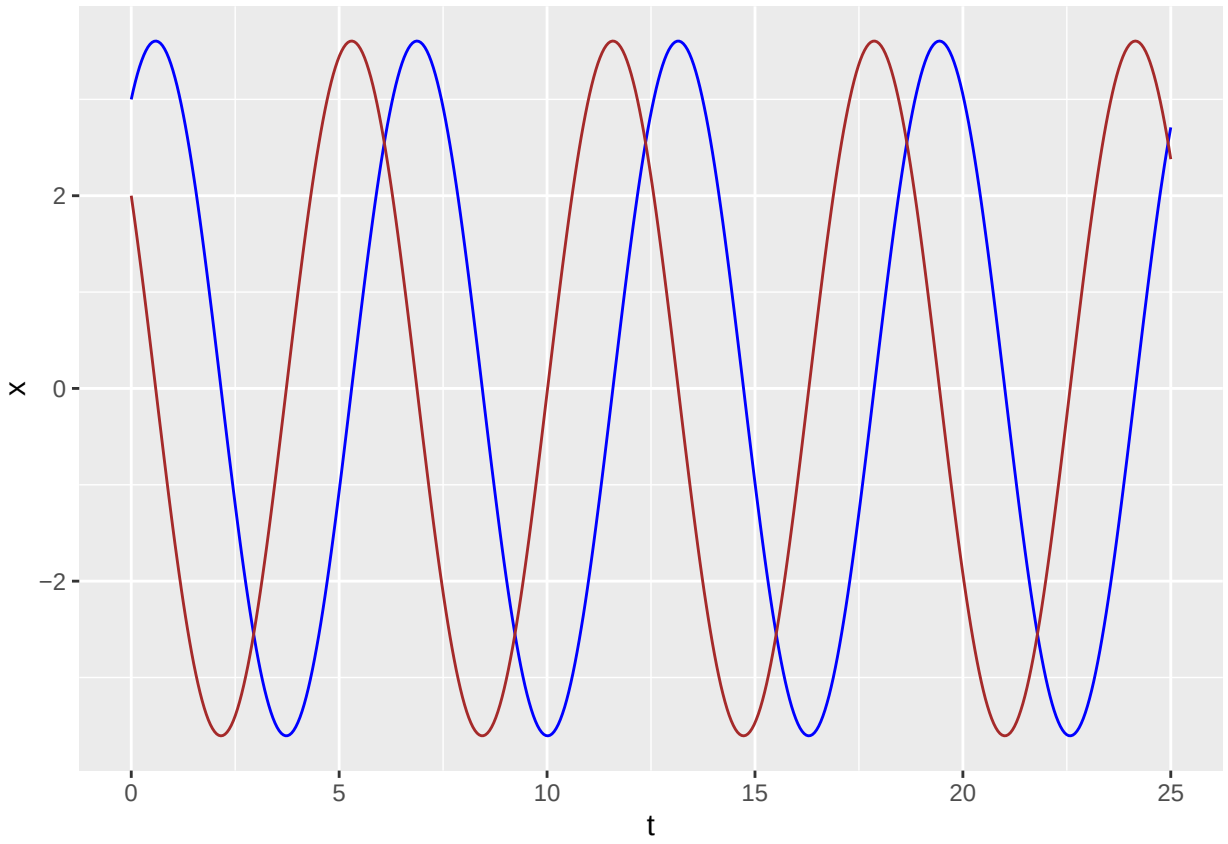
```
my_trajectory <- integrateODE(dx ~ y, dy ~ -x, x=3, y=2, tdur=5)
traj_plot(y(t) ~ x(t), my_trajectory)
```



Note that points on the trajectory are labeled with t.

You can also plot time series with `traj_plot()`

```
my_trajectory <- integrateODE(dx ~ y, dy ~ -x, x=3, y=2, tdur=25)
traj_plot(x(t) ~ t, my_trajectory, color="blue") %>%
  traj_plot(y(t) ~ t, my_trajectory, color="brown")
```



Note: The `my_trajectory` object will consist of a set of functions, the time series for each of the dynamical variables. You can evaluate these functions to find the state at any given time.

```
my_trajectory$x(2)
```

```
## [1] 0.5701543
```

```
my_trajectory$y(2)
```

```
## [1] -3.560186
```

These are mathematical functions of the same sort produced by `makeFun()`, so you can perform whatever operations you want on them in the normal fashion.