16.1 sim.matlab Nonlinear systems in Matlab

Many of the Matlab tools we've used will not work for nonlinear systems; for instance, system-definition with `tf`, `ss`, and `zpk` *and* simulation with `lsim`, `step`, `initial`—none will work with nonlinear systems.

Defining a nonlinear system

We can define a nonlinear system in Matlab by defining its state-space model in a function file. Consider the nonlinear state-space model[1]

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}) \\ &= \begin{bmatrix} x_2 \\ (1 - x_1^2)x_2 - x_1 \end{bmatrix}. \end{aligned} \tag{1}$$

A function file describing it is as follows.

```
type van_der_pol.m
```

```
function dxdt = van_der_pol(t,x)
  dxdt = [ ...
    x(2); ...
    (1-x(1)^2)*x(2) - x(1) ...
  ];
```

Note that `x` is representing the (two) state vector $\mathbf{x}$, which, along with time `t` (t), are passed as arguments to `van_der_pol`. The variable `dxdt` serves as the output (return) of the function. Effectively, `van_der_pol` is simply $f(\mathbf{x})$, the right-hand side of the state equation.

Simulating a nonlinear system

The nonlinear state equation is a system of ODEs. Matlab has several numerical ODE solvers that perform well for nonlinear systems. When

[1] This is a van der Pol equation.

choosing a solver, the foremost considerations are **ODE stiffness** and **required accuracy**. Stiffness occurs when solutions evolve on drastically different time-scales. For a more-thorough guide for selecting an ODE solver, see

mathworks.com/help/matlab/math/choose-an-ode-solver.html.

For most ODEs, the `ode45` Runge-Kutta solver is the best choice, so try it first. Its syntax is paradigmatic of all Matlab solvers.

```
[t,y] = ode45( ...
    odefun, ... % ODE function handle, e.g. van_der_pol
    time, ...   % time array or span
    x0 ...      % initial state
)
```

Details here include

1. the ODE function given must have exactly two arguments: `t` and `x`;
2. the time array or span doesn't impact solver steps; and
3. the initial conditions must be specified in a vector size matching the state vector `x`.

Let's apply this to our example from above. We begin by specifying the simulation parameters.

```
x0 = [3;0];
t_a = linspace(0,25,300);
```

And now we simulate.

```
[~,x] = ode45(@van_der_pol,t_a,x0);
```

Note that since we specified a full time array `t_a`, and not simply a range, the time (first) output is superfluous. We can avoid assigning it a variable by inserting `~` appropriately.

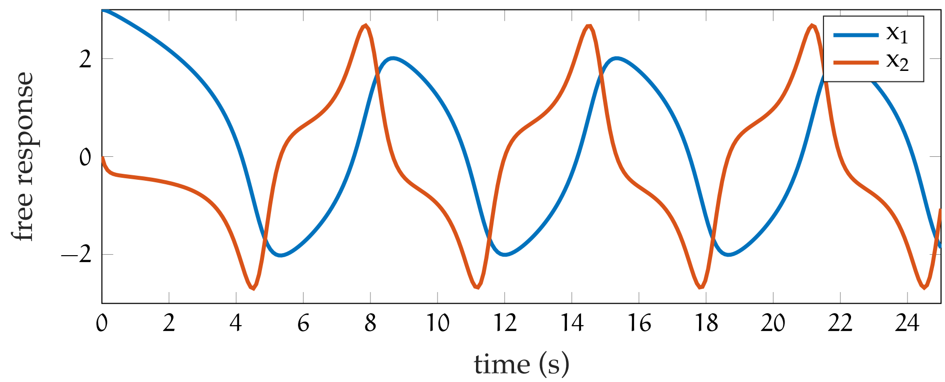

Figure matlab.1: free response plotted through time.

Plotting the response

In time, the response is shown in Fig. matlab.1. Note the weirdness—this is certainly no decaying exponential!

```
figure
plot( ...
    t_a,x.', ...
    'linewidth',1.5 ...
)
xlabel('time (s)')
ylabel('free response')
legend('x_1','x_2')
```

It seems the response is settling into a non-sinusoidal periodic function. This is especially obvious if we consider the phase portrait of Fig. matlab.2.

```
figure
plot( ...
    x(:,1),x(:,2), ...
    'linewidth',2 ...
)
xlabel('x_1')
ylabel('x_2')
```

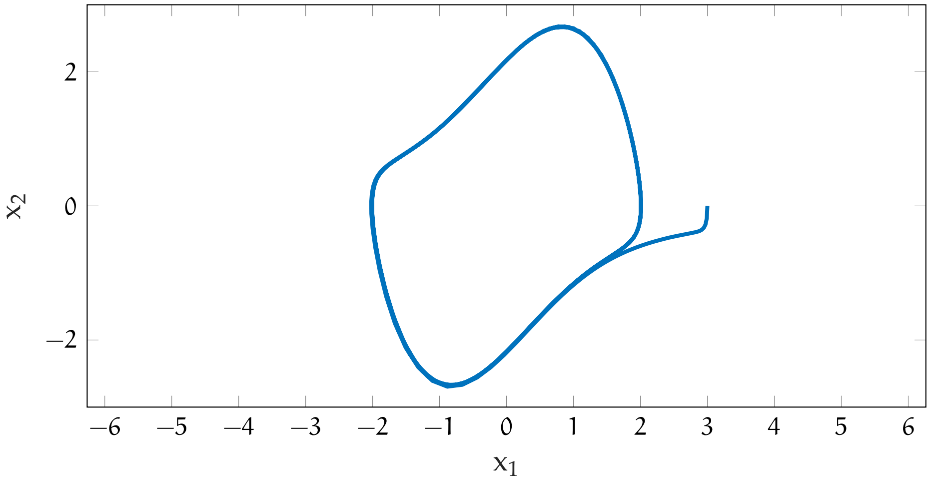

Figure matlab.2: free response plotted in phase space.