

Using MATLAB to simulate systems governed by Linear Ordinary Differential Equations (LODE's)

Brett Ninness

Department of Electrical and Computer Engineering
The University of Newcastle, Australia.

Having now begun a study of how differential equation relationships of the general form

$$\frac{d^n}{dt^n}y(t) + a_{n-1}\frac{d^{n-1}}{dt^{n-1}}y(t) + \dots + a_1\frac{d}{dt}y(t) + a_0y(t) = b_m\frac{d^m}{dt^m}u(t) + \dots + b_1\frac{d}{dt}u(t) + b_0u(t) \quad (1)$$

may be represented and analysed in a theoretical manner, it is also important to gain an initial understanding of how the relation between signals $y(t)$ and $u(t)$ satisfying (1) may be simulated using MATLAB.

1 The `lsim` command

For this purpose, the basic function is `lsim` which is of the following form

```
y = lsim(b,a,u,t);
```

where

$b = [b_m, b_{m-1}, b_{m-2}, \dots, b_1, b_0]$ is a vector of the coefficients specifying the right hand side of (1) which is the differential equation of interest.

$a = [1, a_{n-1}, a_{n-2}, \dots, a_1, a_0]$ is a vector of the coefficients specifying the right hand side of (1).

u = a vector of points which are samples of the signal $u(t)$ that is specified in the differential equation (1).

t = a vector of the *same length* as u . The k 'th element $t(k)$ of t is the time in seconds at which the input sample $u(k)$ is meant to have occurred.

y = a vector of the same length as u and t which represents samples of a signal $y(t)$ which satisfies the differential equation (1). The k 'th element $y(k)$ is the solution $y(t)$ at the time point $t = t(k)$.

The use of this `lsim` function is best illustrated by example.

2 Example of Car Shock Absorber Simulation

In lectures we have seen that the relationship between the road height $u(t)$ and the car height $y(t)$ (above sea level) for a car containing a shock absorber system between wheels and chassis was

$$\frac{d^2}{dt^2}y(t) + \frac{k_d}{m}\frac{d}{dt}y(t) + \frac{k_s}{m}y(t) = g + \frac{k_d}{m}\frac{d}{dt}u(t) + \frac{k_s}{m}u(t). \quad (2)$$

In relation to this, suppose that we are interested in a situation where

Car mass $m = 1000\text{kg}$.

Spring Stiffness $k_s = 2000\text{ N/m}$.

Damper Viscosity $k_d = 500$ Ns/m.

Then this can be specified to MATLAB as follows

```
>> m=1000; ks = 2000; kd = 500;
```

In turn, this allows the co-efficients in the differential equation relationship (2) to be computed as

```
>> a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
```

Note that so far, the constant term g in (2) has been ignored. This is because its only effect is to specify a constant offset to the final solution, which will be clarified later, but is mentioned now to explain why it will not be part of the initial simulations to follow.

To continue, we must now specify the exact experiment that we want to simulate. It is reasonable to expect that the way in which a shock absorber system handles a sudden bump in the road might be of interest. That is, suppose that the road height $u(t)$ signal is a unit step signal $s(t - 1)$ (this would be a 1m high bump!) so that the step actually occurs at $t = 1$ second. Suppose further that we are interested in simulating the resultant $y(t)$ car position signal over a time window of $t \in [0, 10]$ s. Then this situation can be specified via

```
>> t = 0:0.01:10;
```

where the time points have been spaced 10ms (1/100'th s) apart. The unit step at $t = 1$ second with respect to this time scale vector \mathbf{t} can then be specified as

```
u = [zeros(1,100),ones(1,length(t)-100)];
```

Notice the use of the `length` operator (that returns the length of a vector) to ensure that the vectors \mathbf{t} and \mathbf{u} are of the same length.

We are now ready to simulate the shock absorber by dictating the vectors \mathbf{a} and \mathbf{b} that specify the left and right-hand side co-efficients of the differential equation (1)

```
>> a = [1,a1,a0]; b= [b1,b0];
```

and then running the simulation to calculate the car height signal $y(t)$ in the vector \mathbf{y} :

```
>> y = lsim(b,a,u,t);
```

To evaluate this response, let's plot the input bump signal $u(t)$ and the car height response signal $y(t)$ on the same set of axes

```
>> plot(t,y)
>> plot(t,y,'b-',t,u,'r-.')
>> legend('Car height y','road height u')
>> xlabel('Time (s)')
>> ylabel('height')
>> title('Road height and car height')
>> grid on
```

If you have copied the preceding commands into a MATLAB command window, you should now see a plot in front of you that is the same as shown in figure 1. Note that a plot can be generated much more simply than this by avoiding telling `lsim` to pass out the solution vector \mathbf{y} by simply typing

```
lsim(b,a,u,t);
```

In this case, `lsim` automatically generates a plot to communicate the simulation result, but it chooses axes labels automatically rather than the custom ones we have been specifying. Clearly our initial shock absorber design is terrible. The oscillation following a bump is very pronounced, and it is precisely the job of a shock absorber to avoid this sort of response; the car height should settle quickly.

The remainder of this course is aimed at providing insight into responses of systems governed by differential equations. This insight will make it clear why the design flaw shown in figure 1 has occurred and how to fix it.

However, even without this sort of expertise, intuitively it might seem reasonable to increase the 'strength' of the damper; say double it

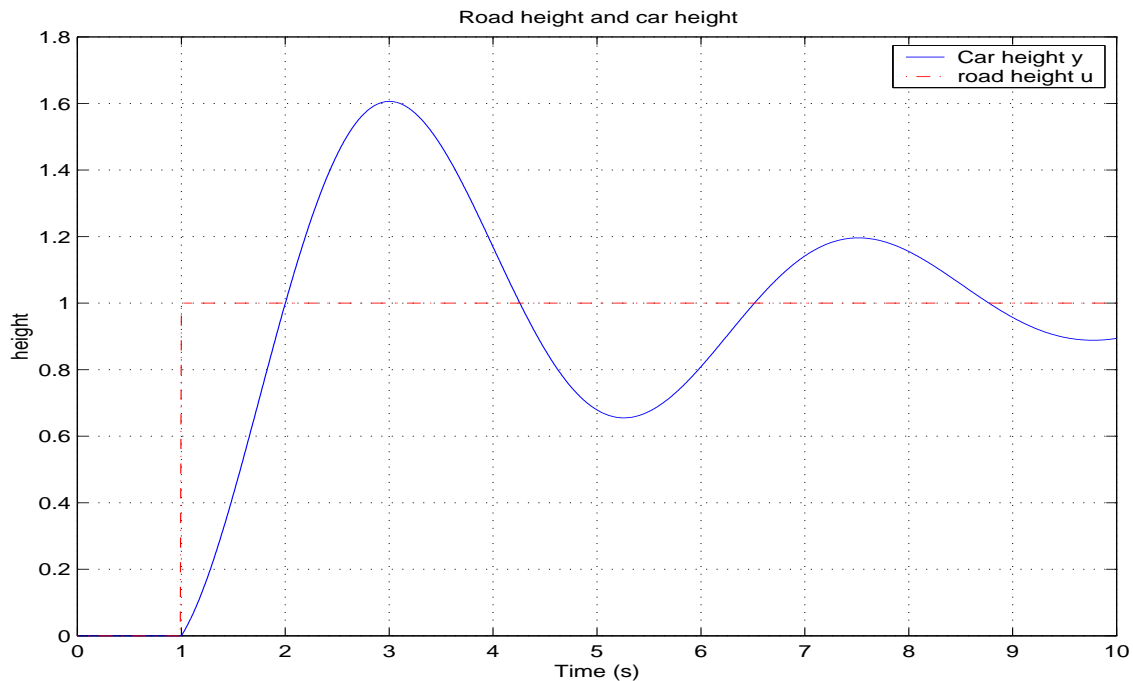


Figure 1: Simulation of shock absorber performance with $m = 1000\text{kg}$, $k_s = 2000\text{ N/m}$, $k_d = 500\text{ Ns/m}$.

```
>> kd = 2*kd
```

```
kd =
```

```
1000
```

and then explore the effect of this by simulation, first by re-calculating the differential equation specification (this would be achieved by using the uparrow key \uparrow to recall the previously typed expressions, there is no need to physically re-type them)

```
>> a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
>> a = [1,a1,a0]; b= [b1,b0];
```

and then re-running the simulation and plotting the results (again, use the uparrow key to repeat these already-typed commands)

```
>> y = lsim(b,a,u,t);
>> plot(t,y,'b-',t,u,'r-.')
>> legend('Car height y','road height u')
>> xlabel('Time (s)')
>> ylabel('height')
>> title('Road height and car height')
>> grid on
```

The results of this simulation experiment are shown in figure 2, and are clearly superior to those shown in figure 1, but still the response appears quite nausea-inducing for passengers. Double again?

```
>> kd = 2*kd
```

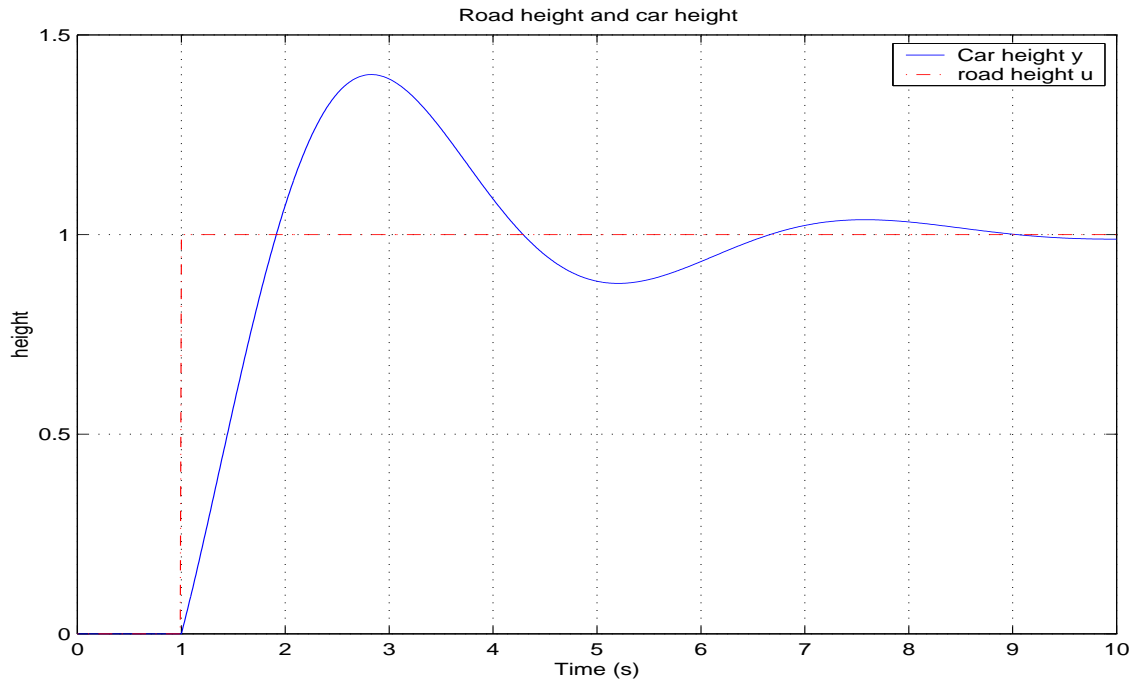


Figure 2: Simulation of shock absorber performance with $m = 1000\text{kg}$, $k_s = 2000\text{ N/m}$, $k_d = 1000\text{ Ns/m}$.

kd =

```

2000
>> a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
>> a = [1,a1,a0]; b= [b1,b0];
>> y = lsim(b,a,u,t);
>> plot(t,y,'b-',t,u,'r-.')
>> legend('Car height y','road height u')
>> xlabel('Time (s)')
>> ylabel('height')
>> title('Road height and car height')
>> grid on

```

The performance of this final design is shown in figure 3, and it now appears acceptable, so we could supply these design parameters of

$$k_s = 2000, \quad k_d = 2000$$

as a specification for a shock absorber suitable for a 1000kg car. However, what if someone tries to mistakenly use this shock absorber on a much larger 2000kg car? Of course, this scenario can be simulated

```

>> m = 2000;
>> a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
>> a = [1,a1,a0]; b= [b1,b0];
>> y = lsim(b,a,u,t);
>> plot(t,y,'b-',t,u,'r-.')
>> legend('Car height y','road height u')
>> xlabel('Time (s)')
>> ylabel('height')

```

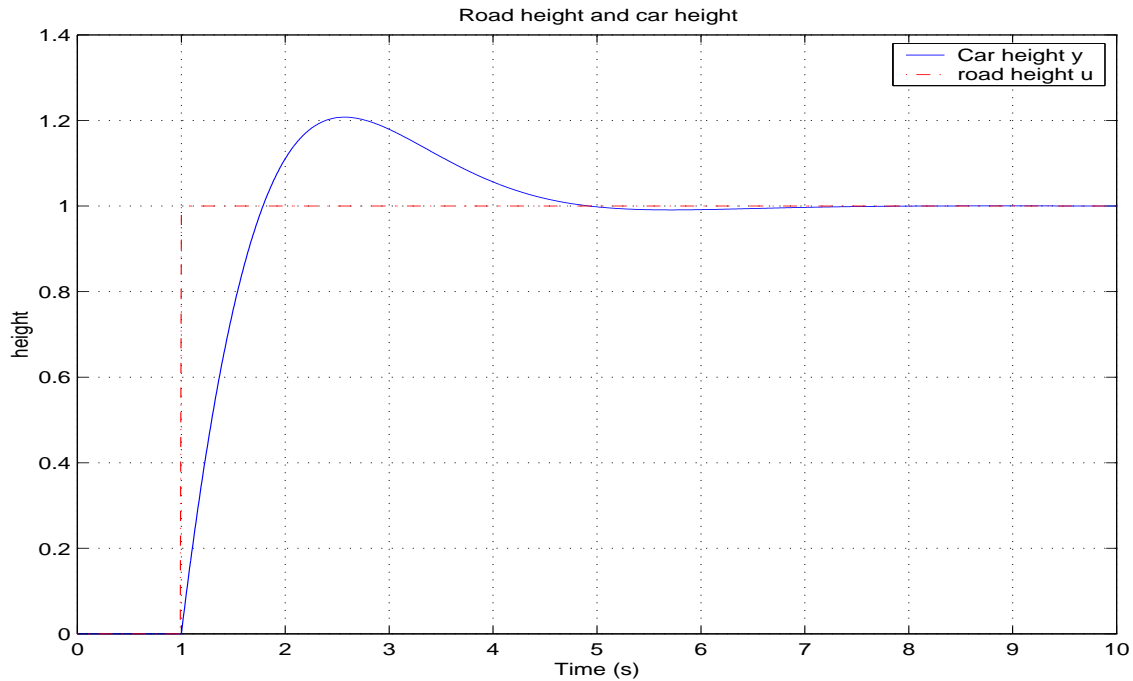


Figure 3: Simulation of shock absorber performance with $m = 1000\text{kg}$, $k_s = 2000\text{ N/m}$, $k_d = 2000\text{ Ns/m}$.

```
>> title('Road height and car height')
>> grid on
```

with the results shown in figure 4. Clearly, if we design a shock absorber for one weight of car, it should not be used on a heavier car without expecting degraded performance.

To conclude this section, we should return to the constant term g which was neglected in (2). In the preceding computer simulation examples, we have in fact been considering the differential equation

$$\frac{d^2}{dt^2}y(t) + \frac{k_d}{m} \frac{d}{dt}y(t) + \frac{k_s}{m}y(t) = \frac{k_d}{m} \frac{d}{dt}u(t) + \frac{k_s}{m}u(t). \quad (3)$$

However, notice that if we have a signal $y(t)$ that satisfies the differential equation (3), then this signal can be used to define a new signal $z(t)$ as

$$z(t) = y(t) + \frac{mg}{k_s}$$

which, via (3) and since the time-derivative of a constant is zero, must satisfy

$$\begin{aligned} \frac{d^2}{dt^2}z(t) + \frac{k_d}{m} \frac{d}{dt}z(t) + \frac{k_s}{m}z(t) &= \frac{d^2}{dt^2}y(t) + \frac{k_d}{m} \frac{d}{dt}y(t) + \frac{k_s}{m} \left[y(t) + \frac{mg}{k_s} \right] \\ &= \frac{d^2}{dt^2}y(t) + \frac{k_d}{m} \frac{d}{dt}y(t) + \frac{k_s}{m}y(t) + g \\ &= \frac{k_d}{m} \frac{d}{dt}u(t) + \frac{k_s}{m}u(t) + g. \end{aligned}$$

That is, if $y(t)$ satisfies the differential equation (3) that ignores the constant term g , then an offset version of this solution $y(t) + mg/k_s$ satisfies the original differential equation (2).

The offset mg/k_s is, in fact, just the height the car would sit at in steady state (for example, when stationary). To see this, set $u(t) = 0$, $du(t)/dt = 0$, $d^2y(t)/dt^2 = 0$, $dy(t)/dt = 0$ in the original

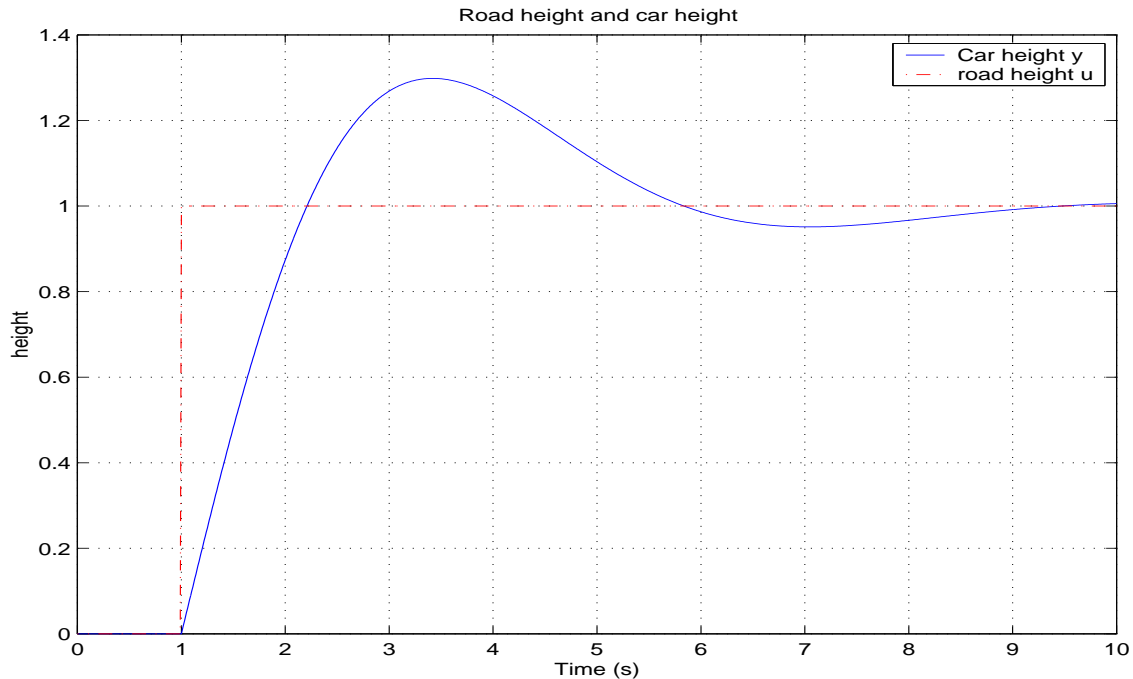


Figure 4: Simulation of shock absorber performance with $m = 2000\text{kg}$, $k_s = 2000\text{ N/m}$, $k_d = 2000\text{ Ns/m}$.

differential equation (3); this implies consideration of a perfectly flat sea-level road together with the car height having reached a non-moving, non-accelerating steady state. Then in this case, from (3)

$$\frac{k_s}{m}y(t) = g$$

which implies $y(t) = mg/k_s$. Neglecting the g term in (3) when forming simulations means that we are simply considering car-height variations around steady state, as opposed to absolute car height including a steady state value.

3 Function files

One obvious feature of the preceding examples (especially for readers who are actually typing the indicated commands into MATLAB) is that there was a common block of operations which needed to be repeated each time a design parameter was changed. Even with the ability to use the up-arrow \uparrow to avoid re-keying, it is still tedious to re-enter these commands.

One solution is to define a new custom function for ourselves which takes as arguments the design parameters k_s , k_d and m together with the specifications u and t of $u(t)$ and then returns a solution vector y defining $y(t)$ while at the same time plotting the response.

That is, we would like to obtain the results of a particular choice of design parameters not by typing in all the following commands used above

```
>> a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
>> a = [1,a1,a0]; b= [b1,b0];
>> y = lsim(b,a,u,t);
>> plot(t,y,'b-',t,u,'r-.')
>> legend('Car height y','road height u')
```

```
>> xlabel('Time (s)')
>> ylabel('height')
>> title('Road height and car height')
>> grid on
```

but rather by some sort of succinct command like

```
>> shock(ks,kd,m,u,t);
```

where `shock` is the name we have made up for our custom function.

Achieving this custom function definition is very easy. Simply open up a text file named `shock.m`, declare it is a function taking arguments in a particular order and returning a vector y by putting the following as the first line of the file

```
function y = shock(ks,kd,m,u,t);
```

and then complete the file with the above commands which were troublesome to repeat. Like so:

```
function y = shock(ks,kd,m,u,t);

a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
a = [1,a1,a0]; b= [b1,b0];
y = lsim(b,a,u,t);
plot(t,y,'b-',t,u,'r-.')
legend('Car height y','road height u')
xlabel('Time (s)')
ylabel('height')
title('Road height and car height')
grid on
```

The command

```
>> shock(ks,kd,m,u,t);
```

will then invoke all the commands in the file `shock.m` and produce the required plot of simulated $y(t)$.

Of course, it is always good to document a function so that on-line help is available. This can be achieved by including the desired on-line help message (commented out using the percent character at the start of the file) like so

```
% y = shock(ks,kd,m,u,t)
%
% Function to calculate shock absorber response where
%
% u,t = vectors specifying road height. That is, u(k) is the
%       road height at time t = t(k)
% ks = spring stiffness constant in N/m
% kd = damper viscosity in Ns/m
% m = mass of car in kg.
%
% written by Brett Ninness, Department of EE & CE
%                               University of Newcastle
%                               Australia.
%                               Last Revised 19/7/2000.
```

```
function y = shock(ks,kd,m,u,t);
```

```

a1 = kd/m; a0 = ks/m; b1 = kd/m; b0 = ks/m;
a = [1,a1,a0]; b= [b1,b0];
y = lsim(b,a,u,t);
plot(t,y,'b-',t,u,'r-.')
legend('Car height y','road height u')
xlabel('Time (s)')
ylabel('height')
title('Road height and car height')
grid on

```

in which case, the query `help shock` will provide the specified help message:

```

>> help shock

y = shock(ks,kd,m,u,t)

Function to calculate shock absorber response where

u,t = vectors specifying road height. That is, u(k) is the
      road height at time t = t(k)
ks = spring stiffness constant in N/m
kd = damper viscosity in Ns/m
m = mass of car in kg.

written by Brett Ninness, Department of EE & CE
      University of Newcastle
      Australia.
      Last Revised 19/7/2000.

```

>>

A final point to note about function files is that, as for other computer languages like Pascal, C, Fortran and so on, variables declared and used within a function are local to that function. For example

```

>> clear
>> t = 0:0.01:10;
>> u = [zeros(1,100),ones(1,length(t)-100)];
>> y = shock(2000,200,400,u,t);
>> who

```

Your variables are:

```
t          u          y
```

will produce the required simulation, but the only extra variable available after running the function `shock` is the vector `y` which was specified to be passed out of the function. The internal (local) variables like `a1`, `b0` are not available.

4 State Space Representations

The final issue for this document is how MATLAB may be used to simulate models specified in state-space form. Firstly, conversion from the differential equation format (1) to the state space format

$$\begin{aligned}\frac{d}{dt}\mathbf{x}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}\mathbf{x}(t) + Du(t)\end{aligned}$$

is achieved by the `tf2ss` ('transfer function to state-space') function. As a specific example, suppose that the differential equation form is specified to MATLAB as follows

```
>> a = [1,0.25,1];  
>> b = [0.25,1];
```

Then the state-space representation is found as

```
>> [A,B,C,D] = tf2ss(b,a)
```

A =

```
   -0.2500   -1.0000  
    1.0000         0
```

B =

```
    1  
    0
```

C =

```
    0.2500    1.0000
```

D =

```
    0
```

Notice that MATLAB does nothing more than use the controller canonical form realisation . This state-space specification can also be used as a specification for `lsim`

```
>> y = lsim(A,B,C,D,u,t);
```

and the result will be precisely the same as if the system had been specified in the differential equation form as

```
>> y = lsim(b,a,u,t);
```

That is, MATLAB is able to intelligently consider the arguments presented to `lsim` in order to determine if a system is being specified to it in state-space or differential-equation form.

5 An Advanced Topic - Abstract System Representations

Since version 5.0 of MATLAB, it has been possible to unify differential-equation and state-space representations of systems into a single data object called a 'system'. This is achieved by using the object oriented features of MATLAB, which are considered here as an advanced topic. For example, to specify a system in differential-equation format (which MATLAB calls 'transfer-function' format), the `tf` constructor

```
>> sys1 = tf(b,a)
```

Transfer function:

```
    0.25 s + 1  
-----  
s^2 + 0.25 s + 1
```

Note from the above, that the resulting data type knows how to print itself out in a nice format. This differential equation form can then be converted to a state space system description using the `ss` operator

```
>> sys2 = ss(sys1)
```

```
a =
```

	x1	x2
x1	-0.25	-1
x2	1	0

```
b =
```

	u1
x1	1
x2	0

```
c =
```

	x1	x2
y1	0.25	1

```
d =
```

	u1
y1	0

Continuous-time model.

Again, the resulting data type (being an object with methods associated with it) knows how to print itself out.

Either of these object-oriented system representations may be used with `lsim`:

```
y = lsim(sys1,u,t);
```

In fact, if one asks for help on `lsim` by typing `help lsim` at the MATLAB prompt, then this is the only form of use that is documented.