

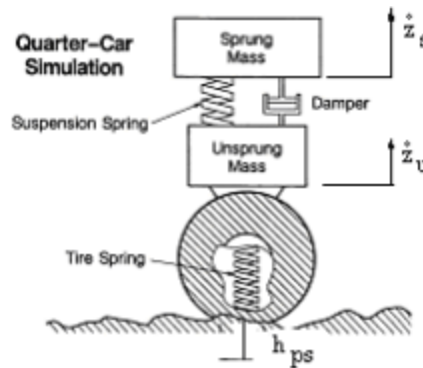
VEHICLE DYNAMICS

Suspension Modelling

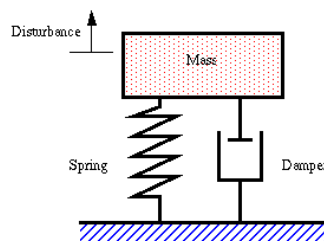
Dr Owen Williams – University of Wales TSD:

Basic Suspension Tutorial:

Suspension systems can be considered as Mass-Spring-Damper systems, which can be modelled using the appropriate equations of motion:



The normal representation of the system being considered is a basic representation of one half of a motorcycle, or indeed one quarter of a car. The level of complexity can vary from model to model depending on the information available, but most situations reduce to differing versions of the basic Mass-Spring-Damper system:



The most basic system is shown above, where we ignore the tyre stiffness and damping properties and consider simply the suspension and the sprung mass:

The characteristic equation of motion for such a system is given by the equation:

$$ma + cv + kx = 0$$

This represents the sum of the forces in the system:

Force = Mass x Acceleration ; $F=ma$

Damping Force = Damping Coefficient x Velocity; $F = cv$

Spring Force = Spring stiffness x Displacement; $F = kx$

This equation can be written in a differential form as:

$$m\ddot{x} + c\dot{x} + kx = 0$$

Or more mathematically:

$$m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = 0$$

The undamped 'circular' natural frequency of such a system is given by:

$$\omega_n = \sqrt{\frac{k}{m}}$$

From the circular frequency, the natural frequency, f_n , can be found by simply dividing ω_n by 2π . Or indeed, without first finding the circular natural frequency, the natural frequency can be found directly using: (This would be your vehicles ride frequency)

$$f_n = \frac{1}{2\pi} \sqrt{\frac{k}{m}}$$

It is normal to use another parameter in the differential equation to represent the damping properties, called the 'damping ratio', usually given by the symbol ζ (called 'zeta'). The damping ratio is given by the equation:

$$\zeta = \frac{c}{c_{crit}} = \frac{c}{2\sqrt{km}}$$

So is the ratio of the damping constant divided by the 'critical damping' value c_{crit}

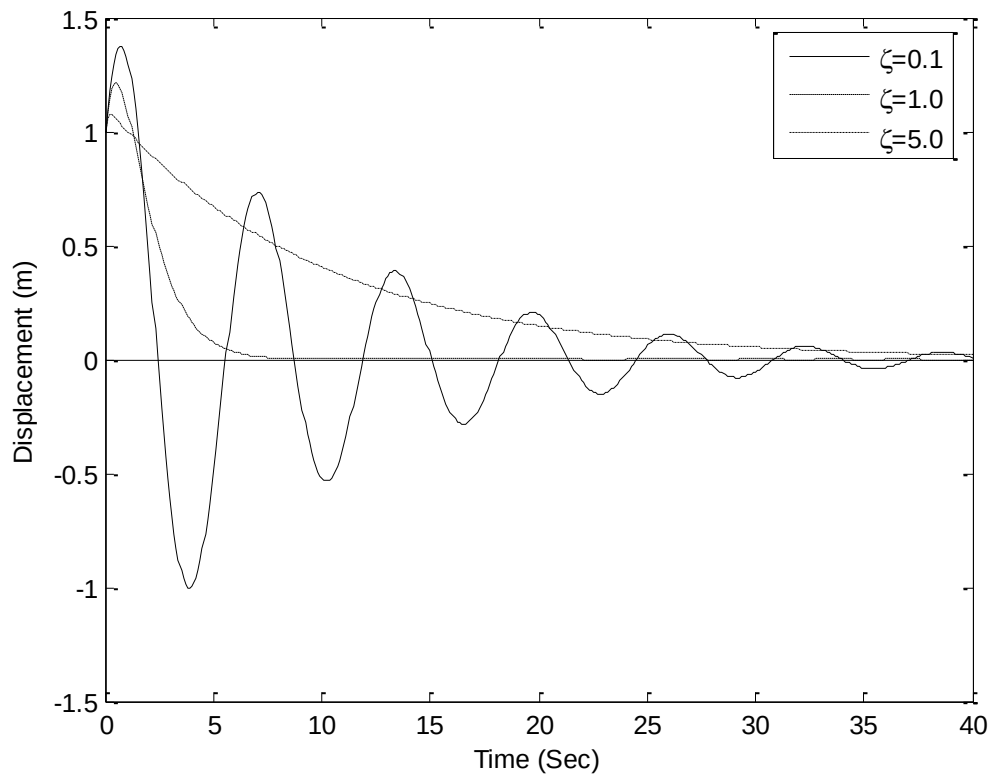
The 'Critical Damping' value is equal to $2\sqrt{km}$



When the system is displaced from its rest position and allowed to move freely, if the system returns to its rest position as fast as possible with no oscillation, the system is said to be critically damped.

When the system is in this critically damped state, it can be seen from the equation that $\zeta = 1$

When the system is under-damped $\zeta < 1$ and when over-damped $\zeta > 1$



MATHEMATICS

Suspension Modelling

Dr Owen Williams – University of Wales TSD:



Basic Mathematical Tutorial: Second Order Linear Differential Equations:

$$a \frac{d^2x}{dt^2} + b \frac{dx}{dt} + cx = f(t)$$

This type of equation is called a second order differential equation, where a , b and c are constants and $f(t)$ is a function of t . Also x is a function of the independent variable, ' t '.

If $f(x) = 0$ then the equation becomes more familiar:

$$a \frac{d^2x}{dt^2} + b \frac{dx}{dt} + cx = 0$$

This is now called a 'Homogeneous Second Order Equation' because the right hand side of the equation is now zero.

To solve an equation of this type, we need to find an x , which is a function of t , that satisfies the equation. Try using the function $x = e^{mt}$ Why??? Well, if you differentiate e^{mt} you get another e^{mt} term.

$$\frac{dx}{dt} = me^{mt}$$

Differentiating again gives:

$$\frac{d^2x}{dt^2} = m^2 e^{mt}$$

So if we substitute this stuff into the homogeneous second order differential equation, we get:

$$am^2 e^{mt} + bme^{mt} + ce^{mt} = 0$$

Factorising out:

$$e^{mt}(am^2 + bm + c) = 0$$

So either ($e^{mt} = 0$) or ($am^2 + bm + c = 0$). For any real values of mt , $e^{mt} \neq 0$ as it is a special property of the exponential function that it cannot be equal to zero. Therefore we must have:

$$am^2 + bm + c = 0$$

Which is a quadratic equation, that we call the '*characteristic equation*' or '*auxiliary equation*'.

This can easily be solved by factorising, or by using the hopefully familiar equation:

$$m = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

There are however, a number of different types of roots to a quadratic equation:

1. Real and Different Roots, m_1 and m_2
2. Real and Equal Roots, $m_1 = m_2 = m$
3. Complex Roots, $m = \alpha \pm i\beta$

The theory of differential equations says that if $x_1(t)$ and $x_2(t)$ are two independent solutions of the differential equation

$$a \frac{d^2x}{dt^2} + b \frac{dx}{dt} + cx = 0$$

Then $A.x_1(t)$ and $B.x_2(t)$ is also a solution (where A and B are any real constants). For the case of *real* and *different* roots, we have:

$$x = Ae^{m_1t} + Be^{m_2t}$$

If the roots are *real* and *equal*:

$$x = (A + Bt)e^{mt}$$

If the roots of the 'auxiliary equation' are complex:

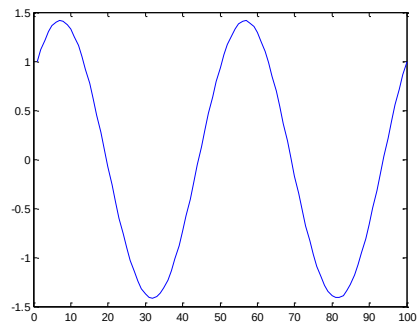
$$x = e^{\alpha t}[A.\cos(\beta t) + B.\sin(\beta t)]$$

There are other general solutions that exist for other special cases of the initial Second Order Differential Equation, for example, for Simple Harmonic Motion of a Mass-Spring system:

$$\frac{d^2x}{dt^2} + k^2x = 0$$

Has the general solution:

$$x = A.\cos(kt) + B.\sin(kt)$$



Which when plotted, shows a clear oscillation which is un-damped.

EXAMPLE

The motion of an un-damped spring-mass system is described by the equation:

$$\frac{d^2x}{dt^2} + 25x = 0$$

Where x is the initial displacement and t is the time. Determine the particular solution for this differential equation with the initial conditions, when $t = 0$, both $x = 1$ and $\frac{dx}{dt} = 10$.

Solution

Since $25 = 5^2$, the differential equation can be written in the form:

$$\frac{d^2x}{dt^2} + 5^2x = 0$$

So, we can use the standard equation for a ODE of this type: $x = A.Cos(kt) + B.Sin(kt)$, with $k = 5$

$$x = A.Cos(5t) + B.Sin(5t)$$

Now substituting the initial conditions, $t = 0$ and $x = 1$

$$1 = A.Cos(5 \times 0) + B.Sin(5 \times 0)$$

$$1 = A(1) + 0$$

$$A = 1$$

To use the other initial condition, $\frac{dx}{dt} = 10$, we need to differentiate first:

$$\frac{dx}{dt} = -5A.Sin(5t) + 5B.Cos(5t)$$

Substituting $t = 0$ and $\frac{dx}{dt} = 10$ gives:

$$10 = -5A.Sin(0) + 5B.Cos(0)$$

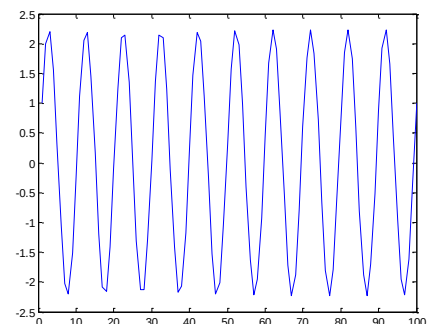
$$10 = 0 + 5B$$

$$B = 2$$

So putting $A = 1$ and $B = 2$ into

$x = A.Cos(5t) + B.Sin(5t)$ gives the solution:

$$x = Cos(5t) + 2.Sin(5t)$$



MATHEMATICS

Suspension Modelling

Dr Owen Williams – University of Wales TSD:



EXAMPLE – Solved using MATLAB :

The previous example shows how the system responds under the specified initial conditions. This example is very easy to solve analytically, but as we develop more complicated systems the analytical solution will become more and more complicated. We will eventually need to start simulation work. It is best to start learning with simple systems so you can really grasp what the software is doing.

There are a number of ways to solve this 2nd Order ODE.

To solve this equation, MATLAB uses a command called 'ODE45' which is an 'Ordinary Differential Equation' solver. There are many other ODE solvers but ODE45 is common for initial value problems.

"The function ODE45 returns the numerical solution to a system of n first-order ordinary differential equations."

But first, we need to reduce the 2nd Order ODE to a First Order ODE. This is done as follows:

If we say that

$$x = x_1 \quad \text{and} \quad \frac{dx}{dt} = x_2$$

Then we find that:

$$\frac{d^2x}{dt^2} + 5^2x = 0 \quad \text{Becomes} \quad \frac{dx_2}{dt} + 5^2x_1 = 0$$

Which is now a first order ODE – So now we rearrange the equation to get:

$$\frac{dx_2}{dt} = -5^2x_1$$

This can now be put into MATLAB:

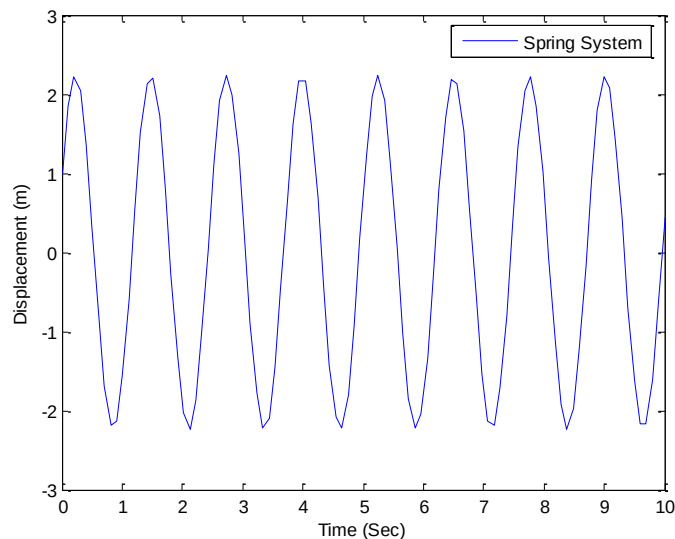
function SM	%Specify use of a function
range=linspace(0,10,100);	%Set the range of values
[t,x]=ode45(@Spring,range,[1,10]);	%ODE Solver with initial conditions
plot(t,x(:,1));	%Specify graph
hold on	%Do not overwrite graph
xlabel('Time (Sec)');	%Format x-axis
ylabel('Displacement (m)');	%Format y-axis
axis([0,10,-3,3])	%Specify x and y axis range
legend('Spring System')	%Draw legend box on graph
function SM=Spring(t,x)	%Name function
SM=[x(2);-25*x(1)];	%Define equation

To get this code into MATLAB:

1. Load the program.
2. Click 'New' icon, or select 'New' and 'M-File'
3. Copy the code into the new window (carefully, as the formatting is important)
4. Save the model (As you would normally save anything)
5. Run the model (Press F5, or select 'Debug' and then 'Run')

MATLAB may ask you if you would like to 'Change Directory'? ... Say Yes...

You should now have a graph appear on screen....



It should be clear that this gives you the same result as that shown earlier for the analytical solve 😊

Notice that the only commands that are solving the equation are the following:

```
function SM=Spring(t,x)                                %Name function
```

This line simply tells MATLAB that we are using a function 'SM' called 'Spring' with variables t and x .

```
SM=[x(2);-25*x(1)];                                    %Define equation
```

The line above tells MATLAB the equation (the 1st order equation that we defined earlier)

```
[t,x]=ode45(@Spring,range,[1,10]);                    %ODE Solver with initial conditions
```

The next line specifies the solver 'ODE45', the thing to solve '@Spring' and also sets the initial conditions as described in the example ' $x=1$ and $\frac{dx}{dt} = 10$ '.

All the other lines of code are just to tell MATLAB to plot the results, what axes to use, and to format the graph as required.

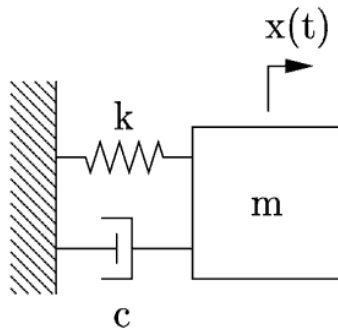
MATHEMATICS

Suspension Modelling

Dr Owen Williams – University of Wales TSD:

Single-Degree-of-Freedom Modelling Tutorial :

Consider a spring -mass-damper system with a mass m , a spring with a spring stiffness k and a damper with a damping coefficient of c .



The mass is subjected to a force $F(t) = x_0 k$ and both gravity and friction are ignored. If x describes the displacement of the system, then the governing equation of motion is of the form:

$$\frac{d^2 x}{dt^2} + 2\zeta \frac{dx}{dt} + x \cdot \omega_n^2 = 0$$

The damping factor ζ (called 'zeta') is given by:

$$\zeta = \frac{c}{2m\omega_n}$$

And the natural frequency is given by:

$$\omega_n = \sqrt{\frac{k}{m}}$$

Solving in MATLAB:

We can now determine the free response of the linear spring-mass-damper system given the initial conditions:

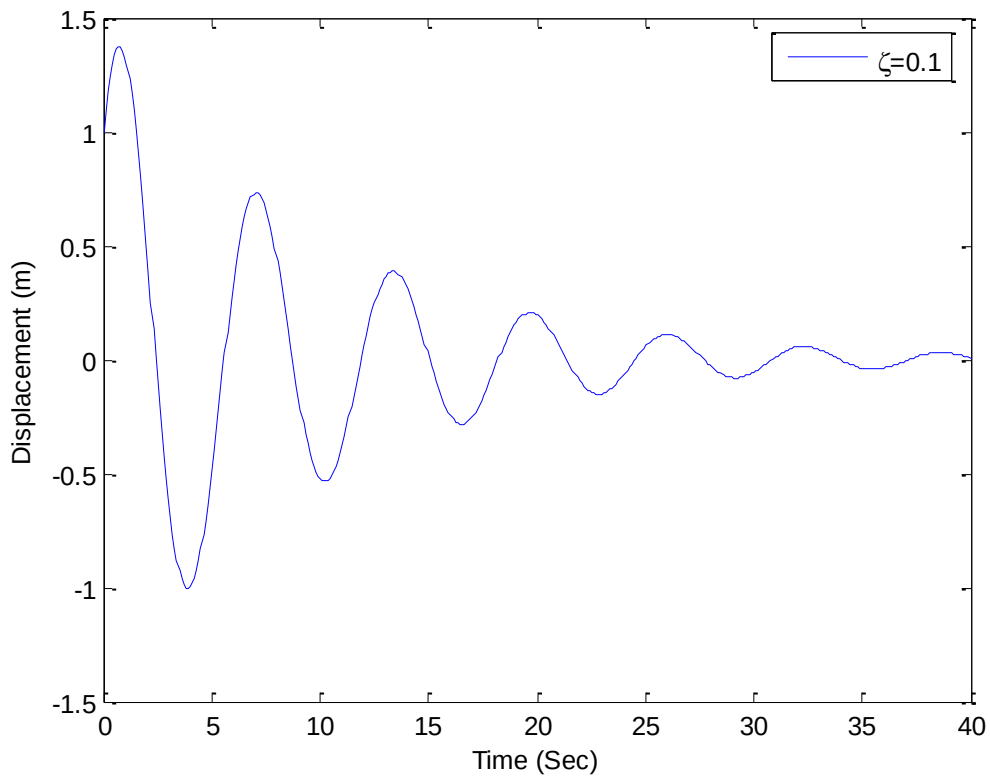
$$x(0) = x_0$$
$$\frac{dx(0)}{dt} = v_0$$

We will also look at three different values of the damping factor ζ . When the value of $\zeta = 0.1$ the system is under-damped, when $\zeta = 1$ the system is said to be critically damped, and when $\zeta = 5.0$ the system is over-damped. The system is set in motion from the initial conditions $x_0 = 1$ and $v_0 = 1$.

To determine the response of the system over the range $0 \leq t \leq 40$ with $\zeta = 0.1$ the script is:

```
function SMD                                     %Specify use of function
zeta=[0.1];                                     %The different damping factor conditions
trange=linspace(0,40,400);                     %The range of time values
[t,x]=ode45(@FreeOscillation,trange,[1,1],[],zeta); %Solver
plot(t,x(:,1));                                %Specify graph properties
hold on                                         %Do not overwrite earlier graphs
figure(1)                                       %Format figure
xlabel('Time (Sec)');                          %Format x-axis
ylabel('Displacement (m)');                   %Format y-axis
axis([0,40,-1.5,1.5])                         %Specify x and y axis range
legend('\zeta=0.1')                           %Draw legend box on graph

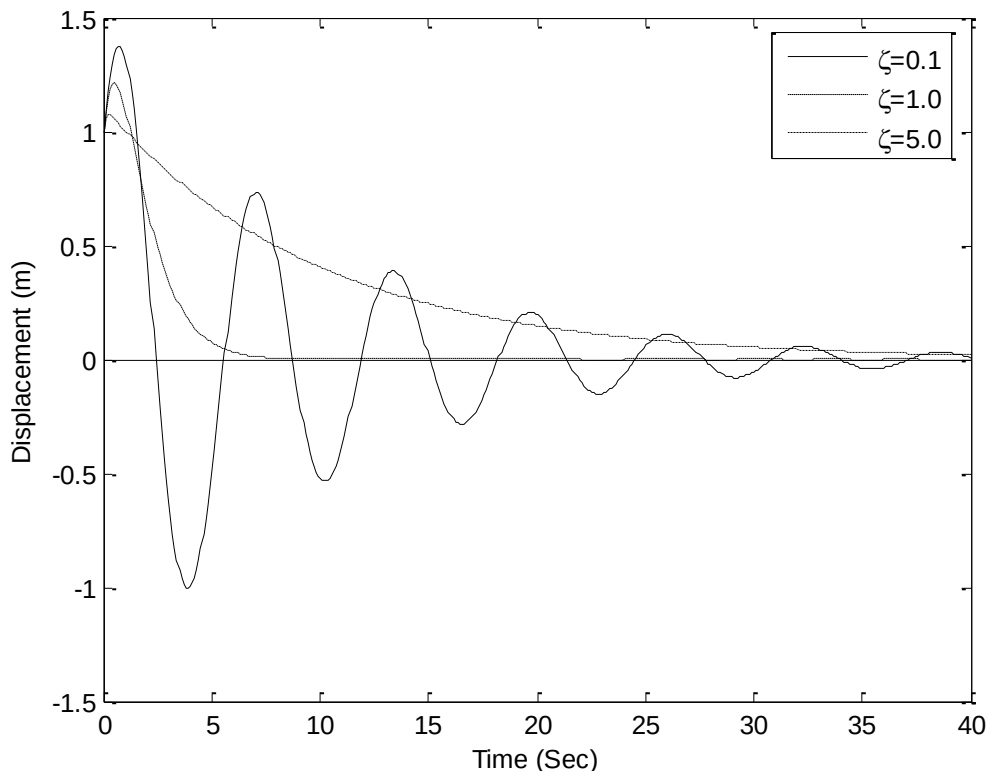
function xdot=FreeOscillation(t,x,zeta)        %Name function
xdot=[x(2);-2*zeta*x(2)-x(1)];                %Define equation
```



We can alter the value of ζ in the above code, or we can write a better version which solves 3 different values of ζ simultaneously and plots the results on the same graph:

function SMD	%Specify use of function
zeta=[0.1,1,5];	%The damping factor conditions
trange=linspace(0,40,400);	%The range of time values
lintyp=char('-k','--k','-.k');	%Specify the line type for graphs
for i=1:3	%Tell MATLAB to run 3 times
[t,x]=ode45(@FreeOscillation,trange,[1,1],[],zeta(i));	%Solver
figure(1);	%Draw graph
plot(t,x(:,1),lintyp(i,:));	%Specify graph properties
hold on	%Do not overwrite earlier graphs
end	
figure(1)	%Format figure
xlabel('Time (Sec)');	%Format x-axis
ylabel('Displacement (m)');	%Format y-axis
axis([0,40,-1.5,1.5])	%Specify x and y axis range
plot([0,40],[0,0],'k-')	%Plot
legend('\zeta=0.1','\zeta=1.0','\zeta=5.0')	%Draw legend box on graph

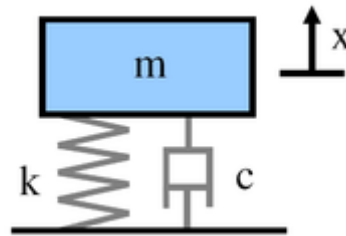
function xdot=FreeOscillation(t,x,zeta)	%Name function
xdot=[x(2);-2*zeta*x(2)-x(1)];	%Define equation
	%Good luck!



The different values of ζ can easily be seen on the above graph, and show the differences between 'critical damping' and both under-damped and over-damped situations.

EXERCISE:

Consider a mass m attached to a spring of stiffness k and a viscous damper with a damping coefficient c .



The displacement, x , is given by:

$$\frac{d^2x}{dt^2} + 2\zeta\omega\frac{dx}{dt} + \omega^2x = 0$$

Where:

$$\omega = \sqrt{\frac{k}{m}}$$

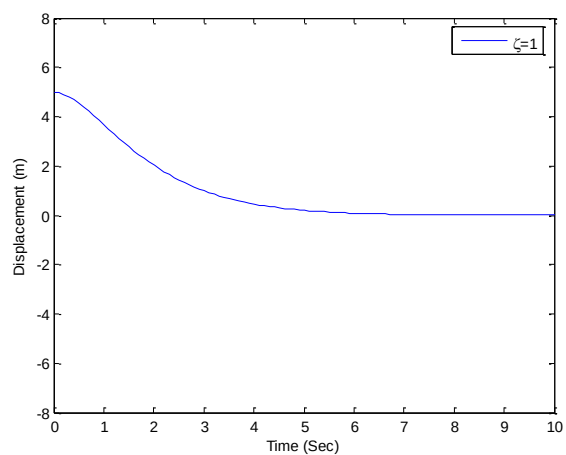
And the viscous damping factor:

$$\zeta = \frac{c}{2m\omega}$$

When $t = 0$, both $x = 5$ and $\frac{dx}{dt} = 0$

Plot the resulting motion when the damping factor $\zeta = 1$.

SOLUTION:



DYNAMICS

Suspension Modelling

Dr Owen Williams – University of Wales TSD:

Modelling a 'real' system :

Consider a typical motorcycle (Or a car, if you're into that kind of thing)

Using sensible values for its mass, weight distribution, suspension stiffness and any other required factors. Try to model the response of the vehicle to a suspension displacement.

You can set the initial conditions, and as such, the 'bump' that excites the system into motion.

Can you show realistic values for the ride frequency of the vehicle?



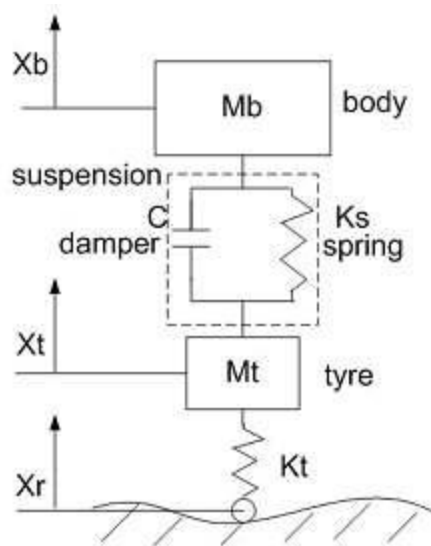
2 DOF MODELS

Suspension Modelling

Dr Owen Williams – University of Wales TSD:

Transfer Function Tutorial :

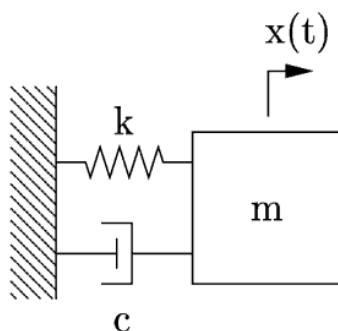
The ODE45 approach for modelling it very useful, but is not powerful enough to deal with compound systems. A compound system is of the form:



For this system outlined above, we have a spring-damper system connecting the un-sprung mass to the sprung mass as is the case for a normal suspension, but we also have a spring connecting the un-sprung mass to the road, representing the tyre stiffness. In later models this will be expanded to include the tyre damping characteristics, but you can already see that this is a significant step forward on the previous mass-spring-damper models. It does however require a new modelling approach.

To understand how these new methods are implemented, we will again look at the simple model previously studied and will model this simple system using both Transfer Function method and State Space method and compare the results with those found earlier.

So, back to the simple model for the moment:



CAUTION: Contains Harder Maths!

The Transfer Function Method:

Notice that the 2nd order ODE equation for the mass-spring-damper system is in the time domain. Meaning it represents things changing in a certain time.

$$m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = f(t)$$

Some analysis however, are easier to perform in the frequency domain. In order to convert to the frequency domain, we need to apply the Laplace Transform to determine the Transfer Function of the system. (See your 1st year Kelvin Sutton Notes!)

The Laplace Transform converts linear differential equations into algebraic expressions which are easier to manipulate. The Laplace Transform converts functions with a real dependent variable (such as time) into functions with a complex dependent variable (such as frequency, often represented by s). Many important characteristics of dynamic or control systems can be determined from the transfer function.

The general procedure to find the transfer function of a linear differential equation is to take the Laplace Transforms of both sides assuming zero conditions, and to solve for the ratio of the output Laplace over the input Laplace. (Perfectly clear?)

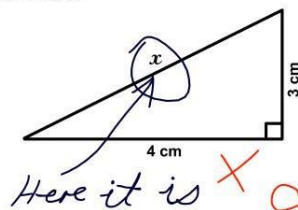
How to find the 'Transfer Function'?

The Laplace Transform allows a linear equation to be converted into a polynomial. The most useful property of the Laplace Transform for finding the transfer function is the differentiation theorem. Several properties are shown below:

	<i>Time Domain</i>	<i>Frequency Domain</i>
<i>Function</i>	$x(t)$	$X(s)$
<i>First Derivative</i>	$x'(t)$	$sX(s) - x(0)$
<i>Second Derivative</i>	$x''(t)$	$s^2X(s) - sx(0) - x'(0)$

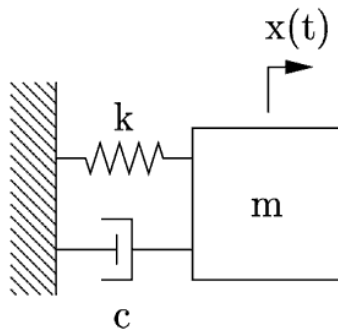
In order to convert the time dependent system equation to the frequency domain, perform the Laplace Transform to the input and output functions and their derivatives. These transformed functions must then be substituted back into the system equation assuming zero initial conditions. Because the transfer function is defined as the output Laplace function over the input Laplace function, rearrange the equation to fit this form.

3. Find x.



Example:

Find the transfer function for the system?



For this system, we know that we have the equation:

$$m \frac{d^2x}{dt^2} + c \frac{dx}{dt} + kx = f(t)$$

Rearrange to get:

$$f(t) - kx - c\dot{x} - m\ddot{x} = 0$$

The notation of the Laplace Transform operation is:

$$\mathcal{L}\{x(t)\} = X(s)$$

$$\mathcal{L}\{x'(t)\} = sX(s) - x(0)$$

$$\mathcal{L}\{x''(t)\} = s^2X(s) - sx(0) - x'(0)$$

$$\mathcal{L}\{f(t)\} = F(s)$$

Now substitute all these terms back into the original equation:

$$F(s) - k[X(s)] - c[sX(s)] - m[s^2X(s)] = 0$$

Which gives:

$$[ms^2 + cs + k]X(s) = F(s)$$

The transfer function is defined as the output Laplace Transform over the input Laplace Transform, and so the transfer function of this second order system is:

$$\frac{X(s)}{F(s)} = \frac{1}{[ms^2 + bs + k]}$$

HOW TO INPUT THE TRANSFER FUNCTION INTO MATLAB

In order to enter a transfer function into MATLAB, the variables must be given numerical values, because MATLAB cannot manipulate symbolic variables without the symbolic toolbox (If you have symbolic toolbox then this is possible too). Enter the numerator and denominator polynomial coefficients separately as vectors of coefficients of the individual polynomials in descending order. The syntax for defining a transfer function in MATLAB is:

transferfunction = *tf*(*num*, *den*)

where *num* is defined as the vector of numerator coefficients, and *den* is defined as the vector of denominator coefficients.

EXAMPLE

Input the transfer function $\frac{X(s)}{F(s)} = \frac{1}{[ms^2 + cs + k]}$ into MATLAB:

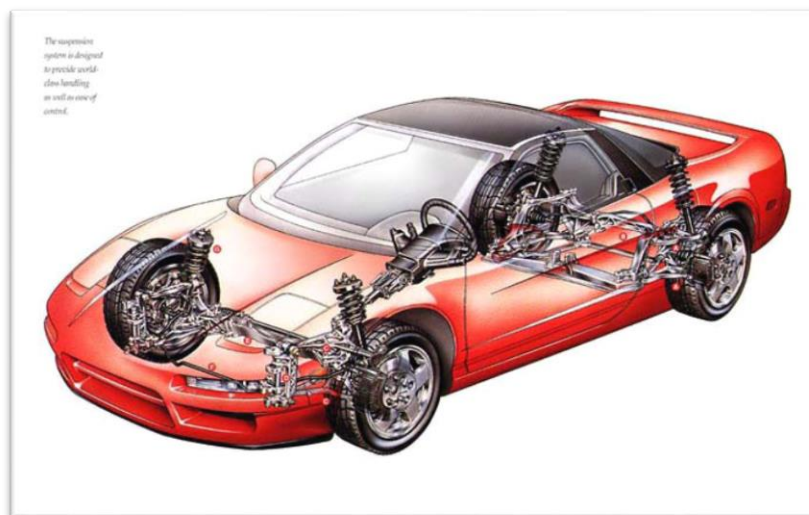
For illustration purposes, this example uses $m = 2$, $c = 5$, and $k = 3$.

```
>> m = 2;  
>> c = 5;  
>> k = 3;  
>> num = [ 1 ];  
>> den = [ m c k ];  
>> tutorial_tf = tf(num, den)
```

MATLAB will assign the transfer function under the name *tutorial_tf*, and output the following:

Transfer function :

$$\frac{1}{[2s^2 + 5s + 3]}$$



Why are we doing this again? ... Suspension! Yes! Still awake?

STEP RESPONSE USING THE TRANSFER FUNCTION

Once the transfer function is entered into MATLAB it is easy to calculate the response to a step input. To calculate the response to a unit step input, use:

```
step(transferfunction)
```

where *transferfunction* is the name of the transfer function of the system.

For steps with magnitude other than one, calculate the step response using:

```
step(u * transferfunction)
```

where *u* is the magnitude of the step and *transferfunction* is the name of the transfer function of the system.

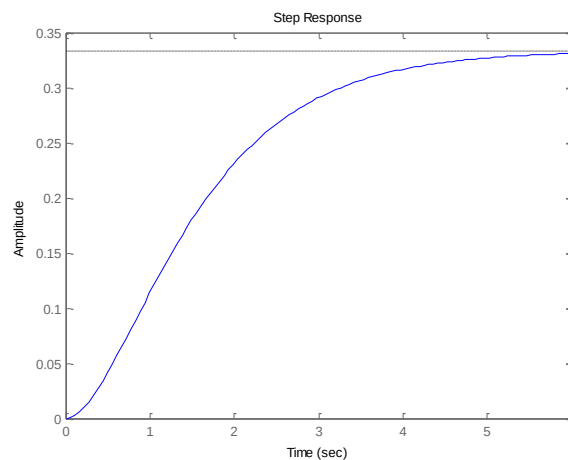
EXAMPLE

Find the unit step response and the step response when $u = 4$ of *tutorial_tf* using MATLAB:

To find the unit step response:

```
>> step(tutorial_tf)
```

The MATLAB output will be the following plot of the unit step response:



This settles to rest in about 6 seconds.

Which is nice... 😊

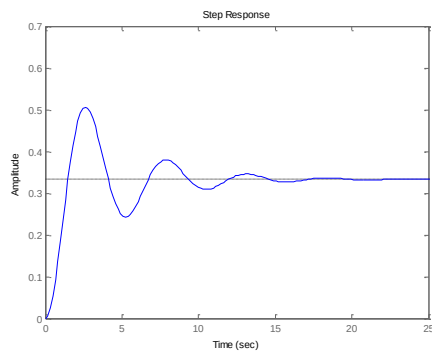
But can we recreate the previous 'under-damped' example by this method?

If we put the previous information into an m-file, we can simply change values as we please to see the system response to different values of mass, damping coefficient and spring stiffness:

Create m-file:

```
m=2          %Mass
c=5          %Damping Coefficient
k=3          %Spring Stiffness
num = [1]    %Numerator
den = [m,c,k] %Denominator
tutorial_tf=tf(num,den) %Transfer Function
step(tutorial_tf) %Step input
```

Now change the damping coefficient to something else... Try $c = 1$



This is clearly now an under-damped system, taking over 20 seconds to settle.

Using your knowledge of suspension, can you now determine the value of c which would give you a system with 'critical damping'?

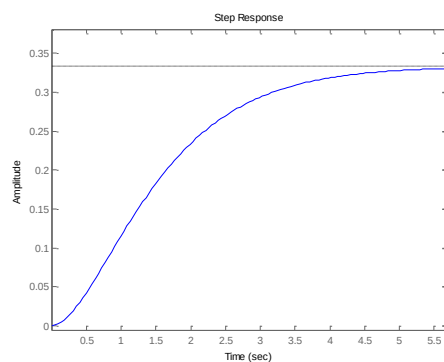
$$c_{crit} = 2\sqrt{km}$$

$$c_{crit} = 2\sqrt{2 \times 3}$$

$$c_{crit} = 4.899$$

Try it...

We can see that this value of c gives the fastest settling time of about 5.5 sec.



MATLAB

Suspension Modelling

Dr Owen Williams – University of Wales TSD:

STATE SPACE Tutorial :



Ok... Large coffee before this one!

The state space model represents a physical system as n first order coupled differential equations. This form is better suited for computer simulation than an n th order input-output differential equation.

The general vector-matrix form of the state space model is:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

Where $y(t)$ is the output equation, and $\dot{x}(t)$ is the state vector.

PARTS OF A STATE SPACE REPRESENTATION

State Variables: a subset of system variables which if known at an initial time t_0 along with subsequent inputs are determined for all time $t > +t_0$

State Equations: n linearly independent first order differential equations relating the first derivatives of the state variables to functions of the state variables and the inputs.

Output equations: algebraic equations relating the state variables to the system outputs.

Note: The state variables are generally variables representing the state of energy storage elements in a system.

The state space model can be found using the following steps:

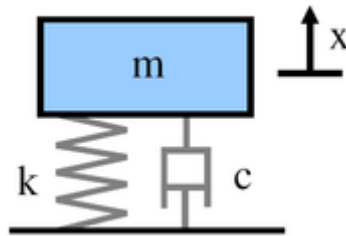
1. *Identify energy storage elements in your system.*
2. *Determine energy relations for each element.*
3. *Pick off state variables from energy relations.*
4. *Find trivial state equations.*
5. *Use physics and algebra to find the other state equations.*
6. *Define a state vector.*
7. *Take derivative of the state vector and relate this derivative to a vector-matrix multiplication of the state vector and the inputs.*



Ok... What on earth does all that mean?

1. IDENTIFY ENERGY STORAGE ELEMENTS AND SELECT STATES

In order to determine your state variables, you must identify energy storage elements. From the system diagram, determine which energy storage elements your problem has. In many cases there can be multiple instances of each element.



Energy Storage Element	Energy Equation	State Variable
<i>Spring</i>	$\frac{1}{2} kx^2$	x
<i>Kinetic Energy</i>	$\frac{1}{2} mv^2$	v
<i>Damper</i>	Does not store energy	<i>None</i>
<i>Applied Force</i>	Does not store energy	<i>None</i>

2. IDENTIFYING TRIVIAL STATE EQUATIONS

Using definitions from mathematics, identify trivial state equations.

For example, by definition:

$$v = x'$$

Therefore the trivial state equation for the above system is $x' = v$ which expresses the derivative of position in terms of states and inputs, in this case, just the velocity.



3. DETERMINING OTHER STATE EQUATIONS USING ELEMENT LAWS AND INTERCONNECTIONS

In order to find the remaining state equations, we have to investigate other relations between the state variables based on element laws and interconnections. The most commonly used of all interconnections is the governing equation from the free body diagram.

EXAMPLE

The equation gathered from the system diagram was:

$$mx'' + cx + kx - f(t) = 0$$

Substituting the definitions of the states into the equation results in:

$$mv' + cv + kx - f(t) = 0$$

Solving for v' gives the state equation:

$$v' = \left(\frac{-c}{m}\right)v + \left(\frac{-k}{m}\right)x + \frac{f(t)}{m}$$

The desired output is for the position, x , so we say:

$$y = x$$

Now the derivatives of the state variables are in terms of the state variables, the inputs, and constants.

$$x' = v$$

$$v' = \left(\frac{-c}{m}\right)v + \left(\frac{-k}{m}\right)x + \frac{f(t)}{m}$$

$$y = x$$



4. PUTTING INTO VECTOR-MATRIX FORM!

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

Our state vector consists of two variables, x and v so our vector-matrix will be in the form:

$$\begin{bmatrix} x' \\ v' \end{bmatrix} = A \begin{bmatrix} x \\ v \end{bmatrix} + Bf(t)$$

$$y(t) = C \begin{bmatrix} x \\ v \end{bmatrix} + Df(t)$$

The first row of **A** and the first row of **B** are the coefficients of the first state equation for x' . Likewise the second row of **A** and the second row of **B** are the coefficients of the second state equation for v' . **C** and **D** are the coefficients of the output equation for y . This yields:

$$\begin{bmatrix} x' \\ v' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1/m \end{bmatrix} f(t)$$

$$y(t) = [1 \quad 0] \begin{bmatrix} x \\ v \end{bmatrix} + [0]f(t)$$

You can see that we are using:

$$A = \begin{bmatrix} 0 & 1 \\ -k/m & -b/m \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 1/m \end{bmatrix}$$

$$C = [1 \quad 0]$$

$$D = [0]$$

5. HOW TO INPUT THE STATE SPACE MODEL INTO MATLAB

In order to enter a state space model into MATLAB, the variables must be given numerical values, because MATLAB cannot manipulate symbolic variables without the symbolic toolbox. Enter the coefficient matrices A , B , C , and D into MATLAB. The syntax for defining a state space model in MATLAB is:

```
statespace = ss(A, B, C, D)
```

where A , B , C , and D are from the standard vector-matrix form of a state space model.

EXAMPLE:

Input the tutorial state space model into MATLAB:

Theoretical values must be determined for the constants m , c , and k . For the sake of this example, we will take $m = 2$, $c = 1$, and $k = 3$ as we have before.

```
>> m = 2;  
>> c = 1;  
>> k = 3;  
>> A = [ 0 1 ; -k/m -b/m ];  
>> B = [ 0 ; 1/m ];  
>> C = [ 1 0 ];  
>> D = 0;  
>> tutorial_ss = ss(A, B, C, D)
```

This assigns the state space model under the name *tutorial_ss* and outputs the following:

```
a =  
      x1  x2  
x1      0   1  
x2 -1.5 -2.5
```

```
b =  
      u1  
x1      0  
x2  0.5
```

```
c =  
      x1  x2  
y1      1   0
```

```
d =  
      u1  
y1      0
```

Continuous-time model.

STEP RESPONSE USING THE STATE SPACE MODEL

Once the state space model is entered into MATLAB it is easy to calculate the response to a step input. To calculate the response to a unit step input, use:

`step(statespace)`

where *statespace* is the name of the state space system.

For steps with magnitude other than one, calculate the step response using:

`step(u * statespace)`

where *u* is the magnitude of the step and *statespace* is the name of the state space system.

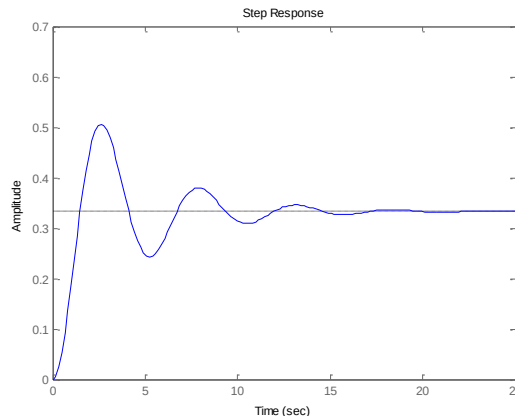
EXAMPLE:

Find the unit step response and the step response when $u = 4$ of *tutorial_ss* using MATLAB:

First to find the unit step response:

```
>> step(tutorial_ss)
```

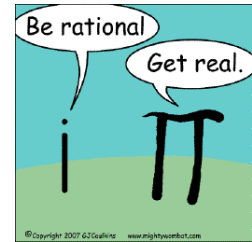
The MATLAB output will be the following plot, and should be quite familiar:



Solution m-file

```
-----  
m=2  
c=1  
k=3  
A=[0 1;-k/m -c/m]  
B=[0;1/m]  
C=[1 0]  
D=0  
tutorial_ss=ss(A,B,C,D)  
step(tutorial_ss)
```

Again, try changing the damping coefficient to 4.899 as before to check the result?



TRANSFER FUNCTION FROM STATE SPACE

MATLAB can determine the transfer function from the state space representation in two ways. To find the numerator matrix and denominator vector of the transfer function of the system from the matrices of the state space model use MATLAB's `ss2tf` command:

```
[num, den] = ss2tf(A, B, C, D)
```

where *num* is defined as the numerator matrix, and *den* is defined as the denominator matrix, and A, B, C, and D are the coefficients of the state space model.

In order to find the entire transfer function system from the state space model, use the following command:

```
transferfunction = tf(statespace)
```

where *statespace* is the name of the state space system, and *transferfunction* is the name of the generated transfer function system.

EXAMPLE:

1. Find *num* and *den*, the numerator and denominator of the transfer function using A, B, C, and D, the state space model coefficient matrices of *tutorial_ss* using MATLAB:

```
>> [num, den] = ss2tf(A, B, C, D)
```

The MATLAB output will be:

```
num =  
    0    0  0.5000
```

```
den =  
 1.0000  2.5000  1.5000
```

2. Find the transfer function system of *tutorial_ss* using MATLAB:

```
>> tutorial_tf = tf(tutorial_ss)
```

MATLAB will assign the transfer function under the name *tutorial_tf*, and output the following:

Transfer function:

```
    0.5  
-----  
s^2 + 2.5 s + 1.5
```

STATE SPACE FROM TRANSFER FUNCTION

MATLAB can also find the state space representation directly from the transfer function in two ways. To find the state space representation of the system from the numerator and denominator of the transfer function

To do this, use MATLAB's *tf2ss* command:

```
[A, B, C, D] = tf2ss(num,den)
```

where *num* is the vector of the numerator polynomial coefficients, and *den* is the vector of the denominator polynomial coefficients.

In order to find the entire state space system in addition to the separate matrices from the transfer function, use the following command:

```
statespace = ss(transferfunction)
```

where *transferfunction* is the name of the transfer function system.

EXAMPLE:

Find A, B, C, and D, the state space vectors of *tutorial_tf* using MATLAB:

```
>> [A, B, C, D] = tf2ss(num,den)
```

The MATLAB output will be:

A =

```
-2.5000  -1.5000  
1.0000    0
```

B =

```
1  
0
```

C =

```
0  0.5000
```

D =

```
0
```

EXAMPLE:

Find the state space system of *tutorial_tf* using MATLAB:

```
>> tutorial_ss = ss(tutorial_tf)
```

MATLAB will assign the state space system under the name *tutorial_ss*, and output the following:

```
a =  
      x1    x2  
x1 -2.5 -0.375  
x2  4      0
```

```
b =  
      u1  
x1 0.25  
x2  0
```

```
c =  
      x1    x2  
y1  0 0.5
```

```
d =  
      u1  
y1  0
```

Continuous-time model.

So how can we use this ↗ ... To help understand this ↘ ... ?



MATLAB

Suspension Modelling

Dr Owen Williams – University of Wales TSD:

Half-Motorcycle or Quarter-Car Modelling :

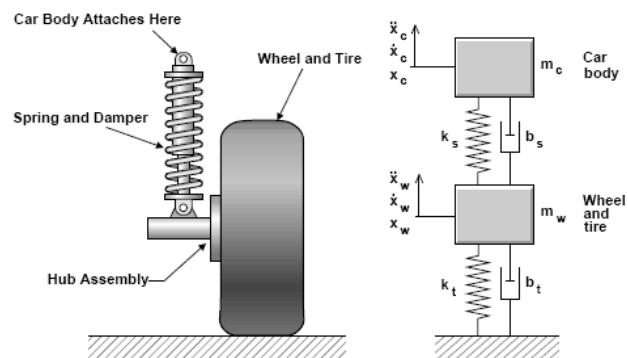
Let us remind ourselves of the characteristic ODE for this system:

$$m \frac{d^2 x}{dt^2} + c \frac{dx}{dt} + kx = f(t)$$

We need to use this ODE again, but with some added complexity:



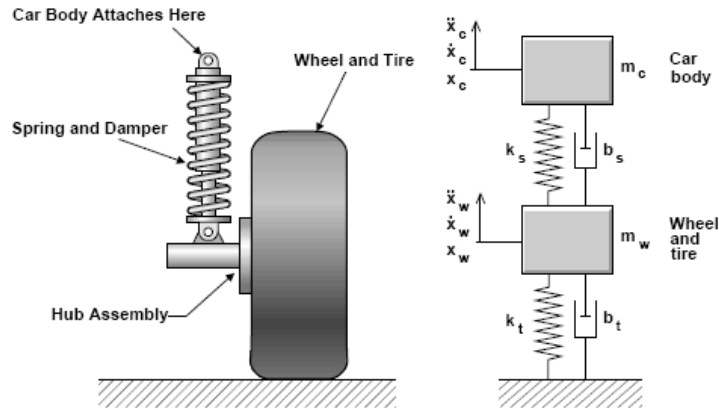
This time we need to consider the suspension stiffness and damping coefficient and also the tyres stiffness and damping coefficient. So we now have 2 masses, two springs and two dampers.



This means that things are going to get much more complicated

Lets derive the required system equations... ☺





From the above diagram we can see that:

We can use the equation:

$$m\ddot{x} + b\dot{x} + kx = 0$$

To get...

$$\dot{x}_t = v_{us} - v_{road}$$

For the tyre velocity..

$$m_{us}\dot{v}_{us} + m_{us}g + k_tx_t + c_t(v_{us} - v_{road}) - k_sx_s - c_s(v_s - v_{us}) = 0$$

For the un-sprung mass..

$$\dot{x}_s = v_s - v_{us}$$

For the sprung mass velocity.. and..

$$m_s\dot{v}_s + m_sg + c_s(v_s - v_{us}) + k_sx_s = 0$$

For the sprung mass

We now need to define the system in vector-matrix form, where:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

WORK THROUGH THIS BIT... and we get:

$$x = [x_t \ v_{us} \ x_s \ v_s]$$

$$y = [v_{us} \ v_s]$$

$$u = [v_{road} \ g]$$

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_t}{m_{us}} & -\frac{c_t+c_s}{m_{us}} & \frac{k_s}{m_{us}} & \frac{c_s}{m_{us}} \\ 0 & -1 & 0 & 1 \\ 0 & \frac{c_s}{m_s} & -\frac{k_s}{m_s} & -\frac{c_s}{m_s} \end{bmatrix} x + \begin{bmatrix} -1 & 0 \\ \frac{b_t}{m_{us}} & -1 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} u$$

$$\dot{y}_{us} = v_{us}$$

$$\dot{y}_s = v_s$$

What in the hell... ??? Don't Panic ... Put it in MATLAB....

Put this Jazz into MATLAB...

```

mus=18                                %Unsprung Mass
ms=150                                %Sprung Mass
ct=1000                                %Tyre Damping Coefficient
cs=1000                                %Suspension Damping Coefficient
kt=80000                                %Tyre Stiffness
ks=20000                                %Spring Stiffness

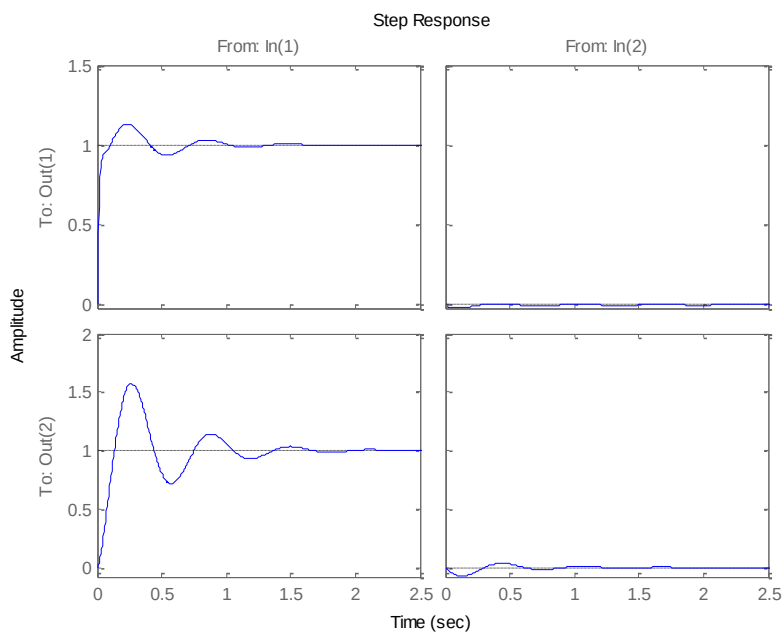
%Specify State Space
A=[0 1 0 0;-kt/mus -(ct+cs)/mus ks/mus cs/mus;0 -1 0 1;0 cs/ms -(ks/ms) -(cs/ms)]
B=[-1 0;ct/mus -1;0 0;0 -1]
C=[0 1 0 0;0 0 0 1]
D=[0 0;0 0]

%Solve State Space
MotoSS=ss(A,B,C,D)

step(MotoSS)                           %Use Step Input
RideF=(1/(2*pi))*sqrt(ks/ms)           %Calculate Ride Frequency
zeta=cs/(2*sqrt(ks*ms))                %Calculate Damping Ratio

```

From this we get:



We also get:

RideF = 1.8378 ... Which is the ride frequency ... $f = 1.8 \text{ Hz}$

Zeta = 0.2887 ... Which is the Damping Ratio ... $\zeta = 0.28$

Utilising Simulink:

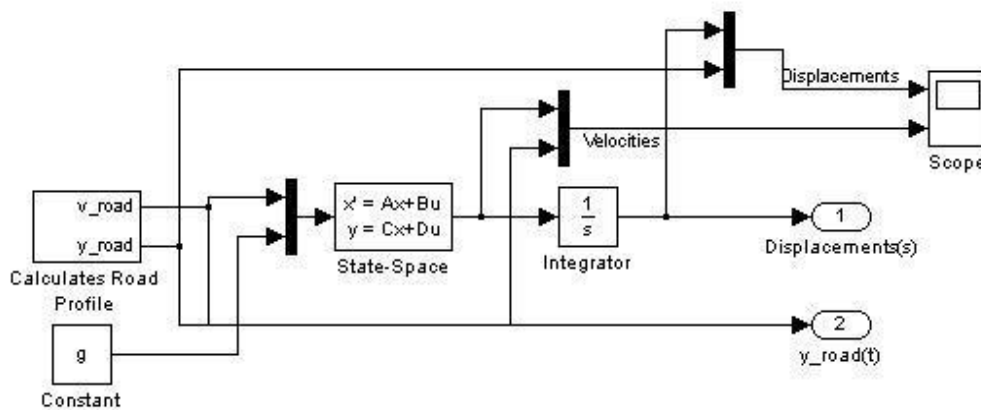
We now want to use the Simulink Environment to set up a working 'Profile' to study how the suspension responds to real 'road inputs' or 'specific bump profiles'.

This is done in the following manner:

Firstly, run the Simulink software:

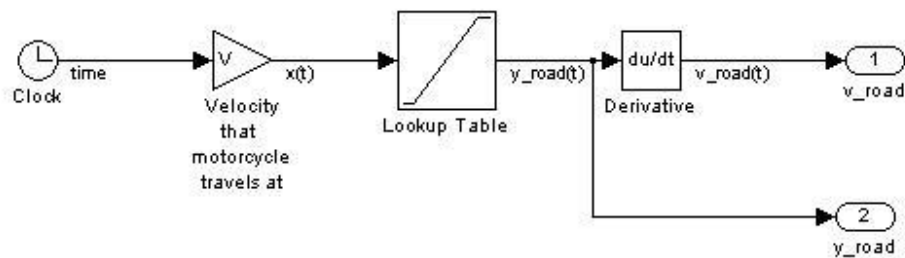
This is achieved by simply typing "Simulink" into the MatLab Command Window:

You will now need to create a 'State Space Solver' in Simulink, and this is done as follows:



This model now has a 'State Space Solver', an input to deal with the position and velocity changes from the road profile and an output for both the resultant displacements and velocities. These outputs are fed back to the MatLab environment to be displayed (Note: The Displacements and Velocities are also displayed using the Simulink 'Scope' block, which will show resultant response – this is good for fault finding.)

We also need to specify the road input profile:



This sets the input velocity of the vehicle and then uses a lookup table to set the profile or bump being used. It then creates an output 'y_road' which is the road position and also the derivative of this signal which is the road velocity change (v_road).

The lookup table can be used to create simple 'step' or 'bump' profiles, but can also be used to create much more complicated road geometry. It could even be used to feed real road data into the model should such data be available to you.

Your final model should create something like this.....

