Vehicle Dynamics and Simulation

TTC066

Lecture Notes (Weeks 1-5)

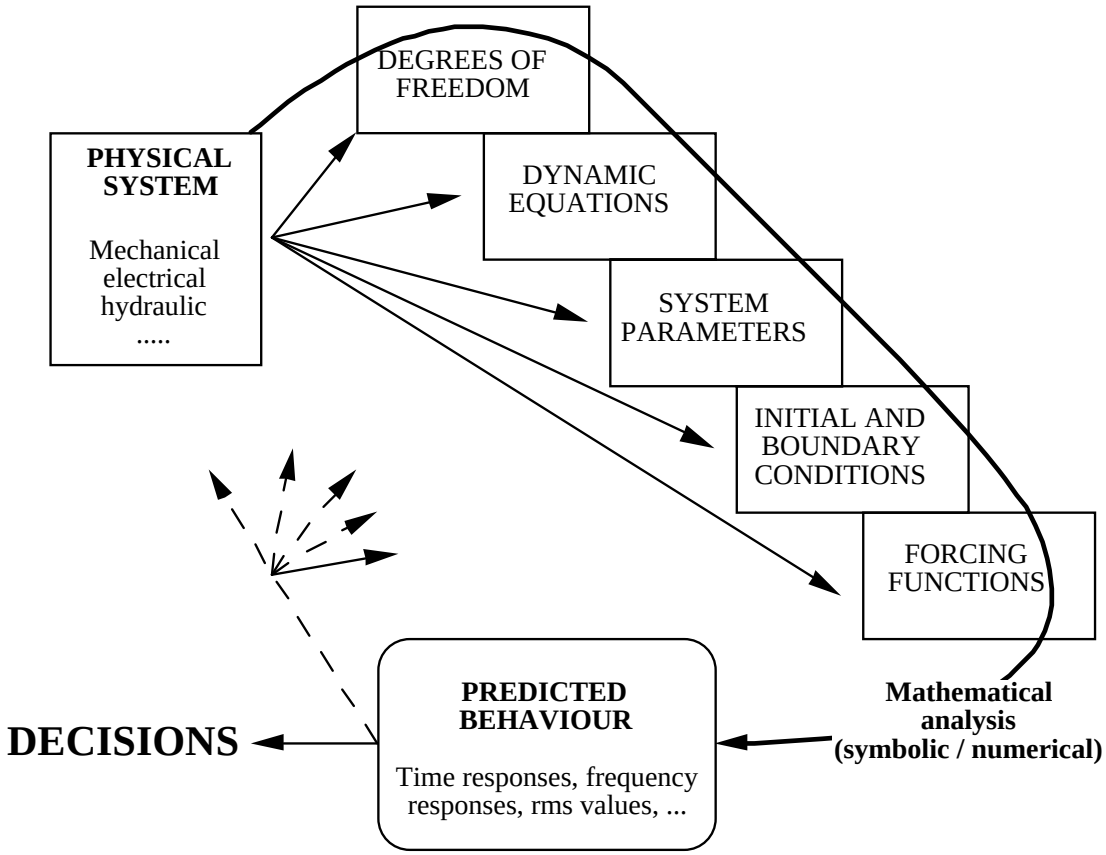
Dr Byron Mason
Dr Matt Best

NB : These notes provide a fairly comprehensive commentary on the course, but should NOT be taken as a replacement for attendance at lectures or computer lab sessions. The course is assessed by coursework alone, and the emphasis is on learning by doing, rather than just by reading !

Contents : **page**

Section 1 : Introduction

The Role of Systems Modelling



Engineering systems are usually modelled as part of a wider process of decision making. Naturally, there are advantages in using models to 'replicate' physical behaviour - for example parametric studies can be undertaken quickly and efficiently (in many cases) for trade-off analysis and optimisation. However there are some subtle factors which should be kept in mind.....

- "All models are wrong - but some are useful" (George Box) - that is, all models have their limitations - so never accept results without question
- "All models are wrong - but then so is vehicle test data" (Matt Best)
- Modelling almost always helps to develop understanding of a system
- The engineer/ modeller decides the valid domain of a model - by careful consideration or by default
- Numerical procedures have their limitations

The differences between a washing machine, an F1 racecar, a rubber ball and a Ford Mondeo

1 **Event 'model'**	2 **Reverse Calculation**	3 **Quasi-static Model**	4 **'Simple' Dynamic Model**	5 **Component Dynamic Model**
Simple sequence of 'steady-state' events; no dynamics, just 'logic', and rules. Eg. If X, then Y A followed by B etc.	Time-stepping through known/required outputs, and using (steady-state) system model (equations / constraints) to calculate required inputs.	Repeated re-calculation of steady-state conditions in forward time, while time-stepping through known inputs.	Integration of differential equations which describe 'reasonably simple' (or simplified) continuously variable dynamic systems.	Differential equations auto-generated from constraints and properties of each individual component
Any computer code. Also MATLAB / STATEFLOW	Any computer code	Any computer code	MATLAB / SIMULINK	Eg ADAMS
Eg. Washing Machine	Eg. Hybrid Powertrain Model meeting speed profile	Eg. Formula 1 lap simuln. Inferior Playstation racing games (ie all except Vanishing Point)	Eg. Rubber ball Superior Playstation racing games (ie Vanishing Point) Ford Mondeo (as modelled eg at Loughborough)	Eg Ford Mondeo (as modelled by Ford)

NB 1-5 are not in order of 'complexity' or 'difficulty' ! 1 = Just a program (washing machine cycle)
2 & 3 = Steady-state modelling (no differential equations) 4 & 5 = Dynamic modelling with differential equations. These need an 'integrater'

Complexity vs. Accuracy ? Viability ? Is more complex always better, or even more accurate ? (Manufacturing/assembly tolerances and wear make it impossible for even the 'perfect' component-based Ford Mondeo model to be really the same as a Mondeo which is out on the street.

Q : Why do Quasi-static methods work well for a Formula 1 racecar, but not (as well) for passenger cars

Section 2a : Modelling and Simulation : Differential Equations

What is a differential equation and how does it relate to the real world ?

We know about 'differentiation', and 'integration' in equations. Eg :

$$y = 3x^2 + 2 \qquad \frac{dy}{dx} = 6x \qquad \int y = x^3 + 2x + c$$

These are what we call analytical functions – where the answer is itself an equation, and you can plug in numerical values for x to give the required value for y, $\frac{dy}{dx}$ or $\int y$. The dynamic behaviour of systems is related using differentiation and integration (with respect to time), but the relationships are more like :

$$M\frac{dv}{dt} = F - \tfrac{1}{2}\rho A C_d v^2 \qquad \text{(i)}$$

ie, with the *derivative of the value dependent on the value itself*. This equation shows a good example – a simplified model of the longitudinal (that's 'forwards' !) dynamics of a car. F is the drive force at the wheels, and v is the forward velocity. The second term on the right hand side represents aerodynamic losses, and M, ρ, A and C_d are (constant) parameters.

How does velocity change over time, if you apply constant engine torque (and hence F *in the model above) to a car ? What are the initial and final values of* $\frac{dv}{dt}$ *and* v *?*

In dynamic simulation, because the dependent variable is (nearly) always time, t, we use the notation $\dot{v}$ in place of $\frac{dv}{dt}$. So, in straight line vehicle acceleration, if position is x, velocity is v and acceleration is a, we have :

$$v = \dot{x} \qquad \text{and} \qquad a = \dot{v} = \ddot{x}$$

Although we can solve equation (i) analytically, the solution will only be valid for a fixed (input) force F, or if F can be replaced by some other (analytical) function of time. And it will give us just one 'result'.

In simulation, we want to generate *time histories* – a trace of how the variable changes over time – as for the 'blue-box' question above. To do this, and allow for maximum flexibility in specifying F, we use *numerical integration*, rather than analytical solution.

Important notation :

variable : Something which changes over time, and which we could find a time history for.

parameter : A constant, which is needed within the model, but which we might vary for separate simulation runs (eg suspension spring stiffness – we might vary this to do a design study).

Generation of differential equations (simple examples)

Dynamic models are described in terms of a series of differential equations which can always be reduced to a set of first order differential equations. Two examples which we will refer to in these notes are the simplified single degree-of-freedom suspension model, and the bouncing ball model (shown below). Note that the 'simplification' stage of model design has already been completed for these cases – the former being reduced to a rigid mass suspended over a point which is fixed on the 'road' by a simple spring and viscous damper (and which moves only vertically); the latter switches between contact and non-contact conditions.

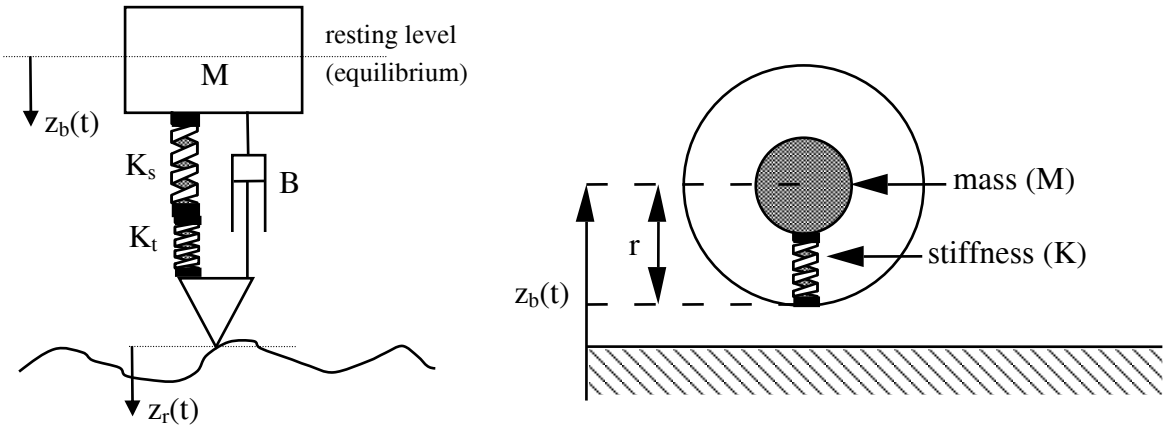


Figure 1 : Suspension (body bounce), and bouncing ball models

(Body bounce) suspension model

$$\frac{1}{K} = \frac{1}{K_s} + \frac{1}{K_t}$$

$$F_s = K\left(z_b - z_r\right) + B_s\left(\dot{z}_b - \dot{z}_r\right)$$

$$\sum F = ma$$

$$-F_s = M\ddot{z}_b$$

$$M\ddot{z}_b = K\left(z_r - z_b\right) + B_s\left(\dot{z}_r - \dot{z}_b\right) \qquad (1)$$

Aside : A quick note on sign conventions

I have drawn the direction I am taking as positive for forces and deflections in the diagrams above (velocities and accels will obviously take the same signs as the deflections). Provided I then write the equations to be consistent with these (note how *F*s depends on $z_b, z_r, \dot{z}_b, \dot{z}_r$) it doesn't matter which direction I take as positive. For example, if forces *Fs* were drawn with arrows into the centre (pulling up on the 'road' and down on the body) I would have to write the equations as

$$F_s = K\left(z_r - z_b\right) + B_s\left(\dot{z}_r - \dot{z}_b\right)$$
$$F_s = M\ddot{z}_b$$

but note that this still gives me the same model.

Note however that when models become more complex and/or use more dimensions (as we will see when we look at vehicle handling) it is much easier to keep everything consistent with a right handed axis system.

Now we reduce this single second order differential equation to a set of first order equations by defining system 'states', x_1, x_2, x_3, etc and inputs u_1, u_2, etc. We seek a form :

$$\begin{aligned} \dot{x}_1 &= f_1(\underline{x},\underline{u}) \\ \dot{x}_2 &= f_2(\underline{x},\underline{u}), \quad \text{etc} \end{aligned} \tag{2}$$

(Note the notation where $\underline{x}$ refers to the whole vector of several states, whereas x_1 refers to just the one state)

We could choose $x_1 = z_b$, $x_2 = \dot{z}_b$, $x_3 = z_r$ and $u = \dot{z}_r$. (I know that choice comes as a bit of a 'rabbit out of a hat' but it comes with experience.) Writing equation 1 in terms of the states and inputs, we get

$$M\dot{x}_2 = K\left(x_3 - x_1\right) + B_s\left(u - x_2\right)$$

so the full set of equations in the form of equation 2 is :

$$\begin{aligned} \dot{x}_1 &= x_2 \quad \text{(from def}^{\text{n}}\text{ of } x_1\text{)} \\ \dot{x}_2 &= \frac{K}{M}\left(x_3 - x_1\right) + \frac{B_s}{M}\left(u - x_2\right) \quad \text{(from dynamic equation)} \\ \dot{x}_3 &= u \quad \text{(from def}^{\text{n}}\text{ of } x_3\text{)} \end{aligned} \tag{2a}$$

This is good (and correct) but better yet is the more cunning choice of states, $x_1 = z_r - z_b$ and $x_2 = \dot{z}_b$ because this obviates the need for a third state (u stays as before). The model is then

$$\begin{aligned} \dot{x}_1 &= u - x_2 \\ \dot{x}_2 &= \frac{K}{M}x_1 + \frac{B_s}{M}\left(u - x_2\right) \end{aligned} \tag{3}$$

Bouncing Ball model (vertical component only)

When the ball is in contact with the ground this is very similar to the suspension, except that the 'road' won't move up and down (I assume) and the reference is the ground, so the sign of +ve z_b is different.

The obvious state choice is $x_1 = z_b$ and $x_2 = \dot{z}_b$, and there is no input, u here. The model is :

if $x_1 < r$,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{K}{M}(r - x_1) - \frac{B}{M}x_2 \end{aligned} \tag{4}$$

else,

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -g \end{aligned} \tag{5}$$

It's interesting that we're modelling mass, M in the case 'if $z_b < r$' (in the K/M and B/M coefficients) but not modelling *weight* in this case. Compare with the 'else' case, where the acceleration is only due to the weight (the $-g$ comes from $F=ma$ where the force is the ball weight).

What would the equations be if we included weight throughout, and how would this change your model results?

In the ball model, there is no input (u) – so we run different scenarios by setting different initial conditions, $x_1(0)$, $x_2(0)$. In the suspension model we can also set initial conditions (and see how the suspension settles) and / or we can define a road input over time, and play this into the model as it integrates. With the full set of information :

1) model

2) initial conditions, $\underline{x}(0)$

3) input(s) over time, $\underline{u}(t)$, $0 < t < T$

we are ready to run a simulation from time $0 < t < T$. Simulink can be persuaded to do this by just pressing the play (►) button, or running **`»sim`** from within Matlab. The subject of the next section of these notes is 'What's happening underneath ?'

Section 2b : Numerical Integration of Ordinary Differential Equations (ODEs)

The simplest method for integrating a set of differential equations

$$\dot{\underline{x}}(t) = f(\underline{x}(t), \underline{u}(t))$$

is the Euler method, which is illustrated in Figure 2(a) :

$$\underline{x}(t+h) = \underline{x}(t) + h\dot{\underline{x}}(t) \tag{6}$$

where h is a small step in time (δt).

This method is quite crude, because the time derivatives $\dot{\underline{x}}(t)$ *continuously* vary over time, yet we are assuming they are constant for the 'small' step h.

Euler is too simple because :

(i) Accuracy can only be ensured by choosing a 'small enough' step size, and thus imposing a large number of function evaluations on the simulation.

(ii) If h is too large, the solution can become unstable as well as inaccurate, with progressively larger values of $\underline{x}$ being predicted.

(iii) The absolute level of accuracy is determined by the model (how rapidly $\dot{\underline{x}}(t)$ varies and whether it includes 'sharp' (switching) nonlinearities, like in the bouncing ball) so it is hard to guarantee a given level of precision.

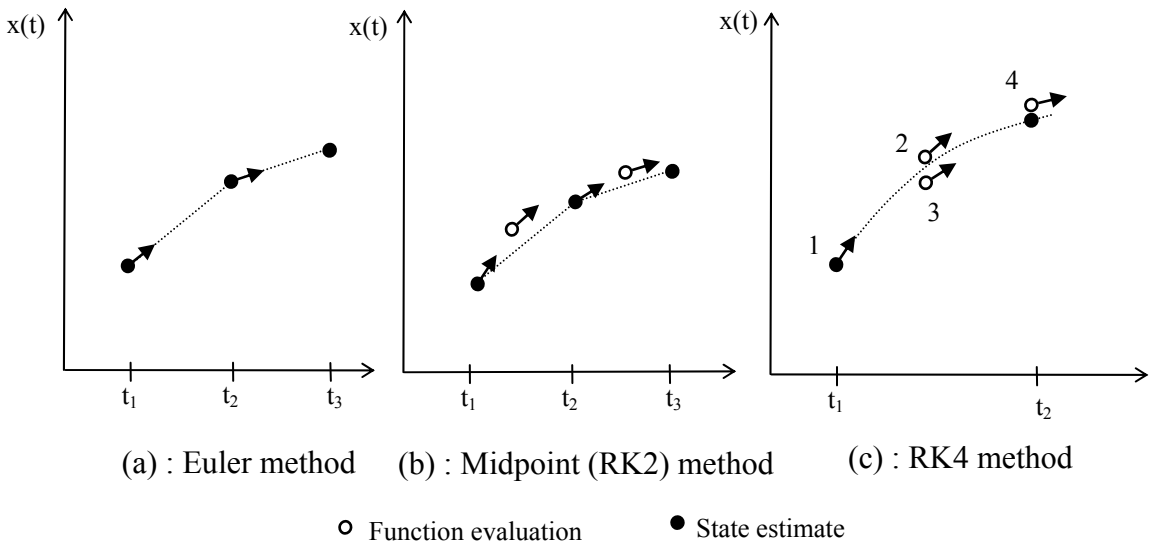


Figure 2 : Time schematic for function evaluations under various integration methods

Figure 2(b) shows a better method, the midpoint method or 'Runge Kutta second order'. Intuitively, if you find the gradient half way along a step, and use this instead of the gradient at the start, you'll get a better estimate of the 'average' behaviour over the time interval :

$$\begin{aligned} \underline{k}_1 &= hf(\underline{x}(t),\ \underline{u}(t)) \\ \underline{k}_2 &= hf(\underline{x}(t)+\underline{k}_1/2,\ \underline{u}(t+h/2)) \\ \underline{x}(t+h) &= \underline{x}(t)+\underline{k}_2+\mathrm{O}(h^3) \end{aligned} \tag{7}$$

The '$O(h^3)$' bit means 'errors of order h^3'. Essentially, the Euler method uses one function ($f(\underline{x}(t),\underline{u}(t))$) evaluation and is accurate to order h^2, the midpoint method uses two (hence second order), and is accurate to $O(h^3)$ etc.

By far the most often used is the fourth order Runge Kutta formula (RK4), which is illustrate in figure 2(c) and is calculated as :

$$\begin{aligned} \underline{k}_1 &= hf(\underline{x}(t),\ \underline{u}(t)) \\ \underline{k}_2 &= hf(\underline{x}(t)+\underline{k}_1/2,\ \underline{u}(t+h/2)) \\ \underline{k}_3 &= hf(\underline{x}(t)+\underline{k}_2/2,\ \underline{u}(t+h/2)) \\ \underline{k}_4 &= hf(\underline{x}(t)+\underline{k}_3,\ \underline{u}(t+h)) \\ \underline{x}(t+h) &= \underline{x}(t)+\frac{\underline{k}_1}{6}+\frac{\underline{k}_2}{3}+\frac{\underline{k}_3}{3}+\frac{\underline{k}_4}{6}+\mathrm{O}(h^5) \end{aligned} \tag{8}$$

This method involves four function evaluations; one at the start, two in the centre and one at the end of the step, and it uses a weighted aggregate to provide the best estimate of the states at $t+h$.

The Runge Kutta methods are better than Euler, because they include 'autocorrection' within a single time step, hence one RK4 step of length h will be more accurate that four Euler steps of length $h/4$.

The graph on the right illustrates this. It shows a section of simulations of the suspension model, settling from an initial deflection of the body.

The 'true states' were calculated using RK4 with a step length of 0.001. RK4, midpoint and euler methods were then tested with equal numbers of function evaluations (RK4 h=0.1, Midpoint h=0.05 and Euler h=0.025)

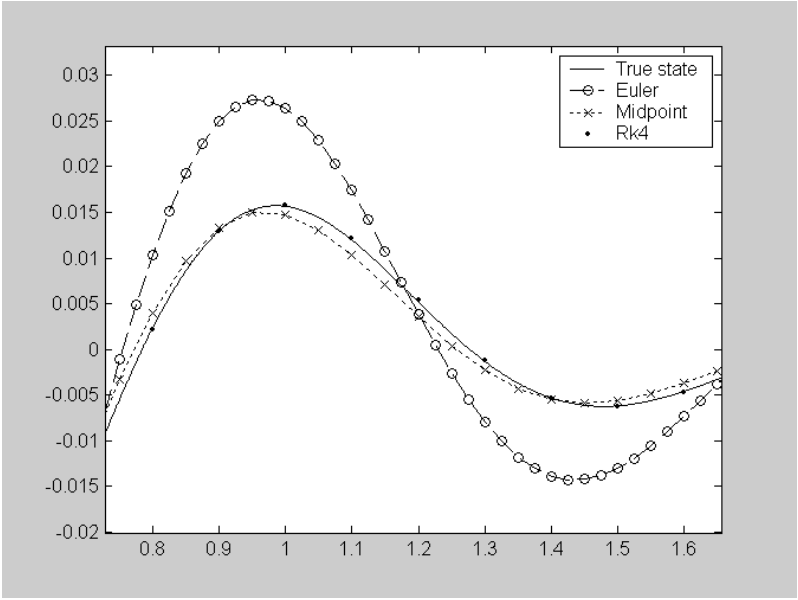


Figure 3 : A 'fair' comparison between integration methods

How do we know what step length we need to achieve a particular level of accuracy though? In general, it isn't appropriate to fix h, and the plot below shows why. The bouncing ball model has a 'sharp' nonlinearity – the model changes suddenly when the ball contacts the floor. In the figure, Simulink has been used to give an 'accurate' simulation of the motion of

the ball after it is released from rest at a height of one metre. The top plot shows vertical height (m) against time, the bottom shows speed (m/s). One dot is given per timestep taken – note that the step length is significantly shortened during the time when the model is switching between contact and non-contact conditions. The other traces are given by fixed step-length RK4 simulations.

For RK4, h=0.1, the time step between function evaluations is too long – the integrator models the ball as in freefall 'just before' it hits the ground, then at the next function evaluation the ball has 'fallen through the floor', so immediately has a very high spring force applied – this explains the subsequent motion, where the ball flies off into the air at great speed (this is slightly unrealistic behaviour). I chose the second case, h=0.03 because for most of the simulation this is the step length the Simulink integrator uses. Similar problems occur again; although it's not as severe, clearly this model is incorrect, as again the simulation shows the ball gaining energy.

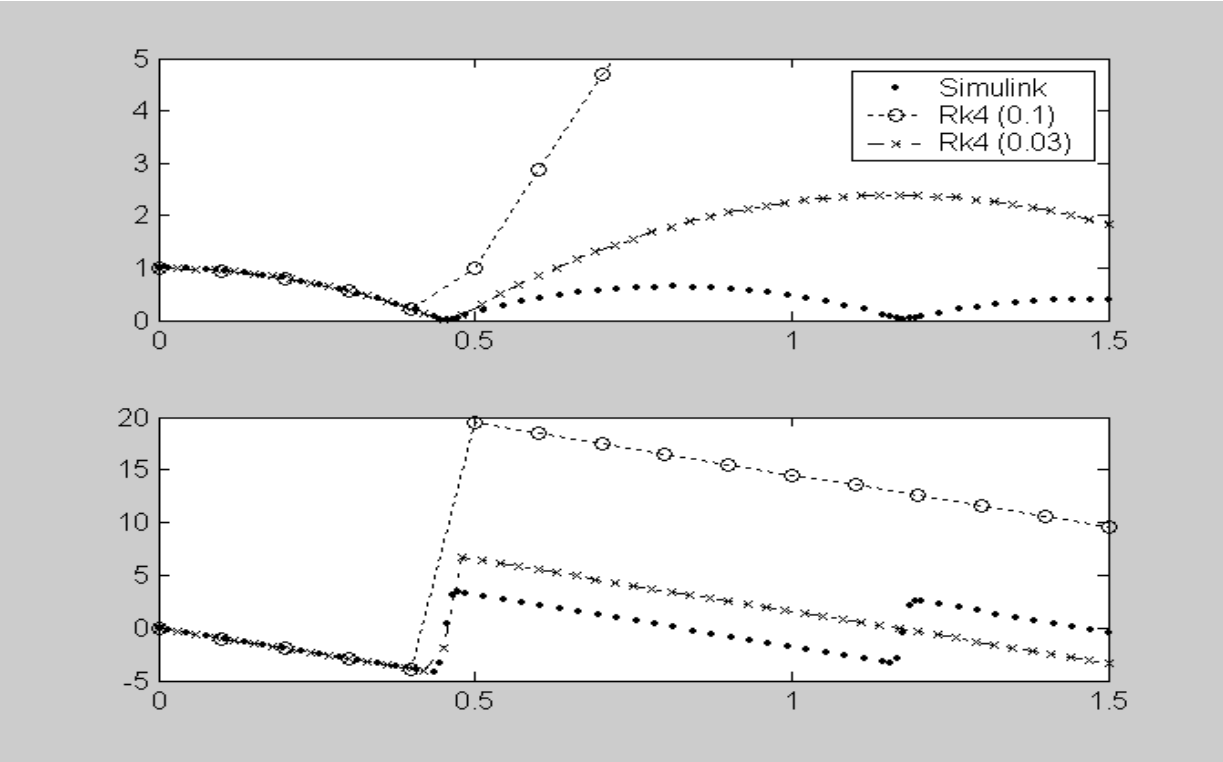


Figure 4 : The ball bounce model uses varying step-length to guarantee accuracy.

It is clear from this example that, however the model is structured, it is essential we have control over the accuracy we are getting. This means integrators have to include an algorithm which automatically adapts the step lengths taken.

Adaptive step size control

"A good ODE integrator should exert some adaptive control over its own progress, making frequent changes in its stepsize. Usually the purpose of this adaptive stepsize control is to achieve some predetermined accuracy in the solution with minimum computational effort. Many small steps should tiptoe through treacherous terrain, while a few great strides should speed through smooth uninteresting countryside. The resulting gains in efficiency are not

mere tens of percents or factors of two; they can sometimes be factors of ten, a hundred, or more." [2]

The simplest way of getting information on accuracy is to perform two small steps (h) and also one double step ($2h$), and compare between the final solutions at $\underline{x}(t+2h)$. This requires a total of 11 function evaluations (not 12 because the first small step shares its first function evaluation with that of the large step). Compared with not making the large step, the 3 extra function evaluations cost 37.5% more computing, but pay dividends in guaranteeing precision, and allowing longer timesteps where possible.

Denoting $\underline{x}_a$ and $\underline{x}_b$ as the one and two step solutions respectively, the *true* value of $\underline{x}(t+2h)$ is given by :

$$\begin{aligned} \underline{x}(t+2h) &= \underline{x}_a + (2h)^5 \varphi + \mathrm{O}(h^6) \\ \underline{x}(t+2h) &= \underline{x}_b + 2\left(h^5\right)\varphi + \mathrm{O}(h^6) \end{aligned} \tag{9}$$

where, to order h^5, the value of φ stays constant over the step. We can now define an error estimate,

$$\varepsilon = \underline{x}_b - \underline{x}_a$$

but note that we can also solve equation 9 to give an improved 'fifth order' estimate for the states,

$$\underline{x}(t+2h) = \underline{x}_b + \frac{\varepsilon}{15} + \mathrm{O}(h^6) \tag{10}$$

The error gives a measure of fourth order accuracy, but the state evaluation actually has fifth order accuracy; this is why adaptive stepsize algorithms, like those in Simulink (see Simulation Parameters in the block diagram menu) refer to their algorithms as RK4/5 (or ODE45 in Simulink).

To adapt the stepsize, consider the order of errors in RK4; ε is of order h^5 (equation 9), so

$$\varepsilon = ch^5$$

where c is some unknown constant. If we desire a particular accuracy level, ε_{des}, this would be related similarly to the step length required to achieve it :

$$\varepsilon_{des} = ch_{des}^5$$

Eliminating c, we have :

$$h_{des} = h\left(\frac{\varepsilon}{\varepsilon_{des}}\right)^{1/5} \tag{11}$$

Thus to guarantee a minimum accuracy ε_{des} we could perform the two small and one big steps at the current best guess steplength h, then use equation 11 to give h_{des}, the stepsize that would have 'just met' the required accuracy over that step. If $h_{des} < h$, the accuracy was not good enough, so the step should be *repeated* with the new h. Conversely, if $h_{des} \geq h$, accuracy was good enough (so there's no need to repeat the step) and we can afford to increase the step length on the next step taken.

The problem is slightly more complicated than it seems though, because, from equation 10, ε is actually a vector (one value per state) so further decisions might be needed; do we use the largest absolute error in equation 11, or is ε_{des} scaled according to expected values of the states (some states may have much larger values than others) ?

Alternatives (particularly for 'Stiff' systems)

In my source for this section of the notes [2], the authors comment that, *'For many scientific users, fourth order Runge Kutta is not just the first word on ODE integrators, but the last word aswell'*, and this is true for most applications that you might use (you'll rarely need to change the integrator away from ODE45 in Simulink). Be aware that there are alternatives though, with exotic names like Bulirsch-Stoer 'predictor-corrector' and Runge-Kutta Fehlberg 'local-extrapolation' (the latter has largely superceded RK4 as described in these notes – and is probably at the heart of Simulink's ODE45 – but the principles are very similar).

The major 'alternative' which you may actually *need* for certain types of model, are the class of integrators designed for 'stiff' systems – that is, systems which have some relatively slow dynamics and some very fast dynamics. A practical example might be in the simulation of a suspension system, where the wishbone mounting bush forces are dynamically modelled as high rate springs, along with the main spring and damper. A simple mathematical example of a stiff system is :

$$\dot{x}_1 = 998x_1 + 1998x_2$$
$$\dot{x}_2 = -999x_1 - 1999x_2$$

which has analytical solution :

$$x_1 = 2e^{-t} - e^{-1000t}$$
$$x_2 = -e^{-t} + e^{-1000t}$$

Here, very small step sizes, $h << 1/1000$ are needed to deal with the e^{-1000t} term, yet this term has negligible effect on the solution unless *t* is very small. Interestingly, it isn't sufficient to just allow for very small *h* near to *t=0*; the use of RK4 on such a system leads to instability because of the nature of the model. 'Poor conditioning' in the model is another way of referring to the problem – $\dot{x}_1$ and $\dot{x}_2$ are very sensitive to a particular combination of errors in x_1 and x_2.

The solution is to use 'implicit' methods, where the model is rearranged such that the state estimate at time $t+h$ is calculated in terms of the derivative at $t+h$, not the derivative at t. A simple example is

$$\dot{x}(t) = -cx(t)$$

where *c* is again some constant. The explicit (or *forward differencing*) Euler method uses,

$$x(t+h) = x(t) + h\dot{x}(t) = (1 - ch)\,x(t)$$

The implicit Euler method uses *backward differencing*, such that

$$x(t+h) = \frac{x(t)}{(1+ch)}$$

There is an overhead in the equation rearrangement (which is automated by the solver, not performed by the model designer), so implicit methods are not used for all systems, but they do provide guaranteed stability, and hence allow numerical solutions for 'stiff' systems. The stiff solvers in Simulink are denoted with suffix 's' in their name (eg ODE15s) and of course, these use more elaborate algorithms that Euler !

Handy Matlab

In the examples shown in figures 3 and 4 above, I wrote some Matlab script files to run the integrators. An example of this code is given below :

```
% Matlab script to perform integration using a 'for' loop
% to step through time.  Uses RK2 (midpoint) method to
% simulate 'body bounce' example (two state version), from an
% initial condition (0.1m suspension extension, zero body velocity),
% assuming zero input.

K = 17647;
M = 400;
B = 1500;


h = 0.05;
tm = [0:h:3]';
n = length(tm);
xm = zeros(n,2);
x = [0.1, 0];
xm(1,:) = x;
for i=2:n,

   % first derivative based on x at start of step :
   xdot(1) = 0 - x(2);
   xdot(2) = K/M*x(1) + B/M*(0 - x(2));
   k1 = h*xdot;

   % then recalculate based on x at midpoint :
   xmid = x + k1/2;
   xdot(1) = 0 - xmid(2);
   xdot(2) = K/M*xmid(1) + B/M*(0 - xmid(2));
   k2 = h*xdot;

   x = x + k2;
   xm(i,:) = x;
end;
```

Section 2c : Linearity and State Space

Linearity

You may have come across this in Control and/or other modules. It is a very important concept which needs to be kept in mind whenever you analyse a system and its responses. The good news is that if a system is linear, there are a number of very useful tools you can use to interpret its behaviour, and you can simulate (and control !) it very easily. But, most systems aren't actually linear in practice. This isn't the disaster it sounds though, because many systems can usefully be approximated as linear (particularly within one or more limited ranges of operation).

A system is linear if its outputs scale (linearly) with its inputs – ie for a linear system, if input $u_1(t)$ causes output $y_1(t)$, and u_2 causes y_2, ie

If the system is linear, and :

$$u_1(t) \rightarrow y_1(t)$$
$$u_2(t) \rightarrow y_2(t)$$

then

$$u_1(t) + u_2(t) \rightarrow y_1(t) + y_2(t)$$
$$cu_1(t) \rightarrow cy_1(t)$$

will be true (for any constant c).

The (state) differential equations (ODEs) for such a system can always be written as linear combinations of the states and inputs, ie :

$$\dot{x}_1 = a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + b_{11}u_1 + b_{12}u_2 + \cdots$$
$$\dot{x}_2 = a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + b_{21}u_1 + b_{22}u_2 + \cdots$$

etc.

where all the a_{ij} and b_{ij} are constants. There are no terms in x_2, no $\sin(x_i)$ or any other function, no saturations, and no lookup tables for $\dot{x}$ as a function of x (unless you're dumb enough to use a lookup table with a straight line relationship !).

The suspension model in Section 2a is linear, but the bouncing ball model is not – although it is linear within each of the two conditions it switches between, you can't use either of these linear models for a whole simulation, so there is little benefit in analysing its response under either linear model separately. Useful linear analysis can only be conducted on so called 'linear time-invariant' (LTI) systems.

State Space

Unlike outer space, state space is not a mystical, far away phenomenon. In the main it refers to LTI models, written in a matrix form :

$$\begin{aligned} \dot{\underline{x}} &= A\underline{x} + B\underline{u} \\ \underline{y} &= C\underline{x} + D\underline{u} \end{aligned} \tag{12}$$

(oooh, vectors and matrices, now I am confused – don't be) Ignoring the second equation for now, the first is just an efficient way of storing a whole (LTI) model in two matrices. For example, if we write out the suspension model of equation 3, but we include a coefficient for every possible state and input, and put them in order, we have :

$$\begin{aligned} \dot{x}_1 &= 0x_1 - 1x_2 + 1u_1 \\ \dot{x}_2 &= {}^{K}\!/_{M}\, x_1 - {}^{B_s}\!/_{M}\, x_2 + {}^{B_s}\!/_{M}\, u_1 \end{aligned}$$

and the state space model *A*, *B* is given by :

$$\dot{\underline{x}} = \begin{pmatrix} 0 & -1 \\ {}^{K}\!/_{M} & -{}^{B_s}\!/_{M} \end{pmatrix} \underline{x} + \begin{pmatrix} 1 \\ {}^{B_s}\!/_{M} \end{pmatrix} u$$

(A) *(B)*

(it's a bit like reverse-population of a matrix multiplication problem, working from the answer, backwards). There are two states, so *A* (which is always square) has size 2x2, and only one input (I haven't underlined *u* in the model), so *B* is 2x1.

The alternative (longer) form for the same model was described in equation 2a. In 'longhand' this becomes :

$$\begin{aligned} \dot{x}_1 &= 0x_1 + 1x_2 + 0x_3 + 0u_1 \\ \dot{x}_2 &= -{}^{K}\!/_{M}\, x_1 - {}^{B_s}\!/_{M}\, x_2 + {}^{K}\!/_{M}\, x_3 + {}^{B_s}\!/_{M}\, u_1 \\ \dot{x}_3 &= 0x_1 + 0x_2 + 0x_3 + 1u_1 \end{aligned}$$

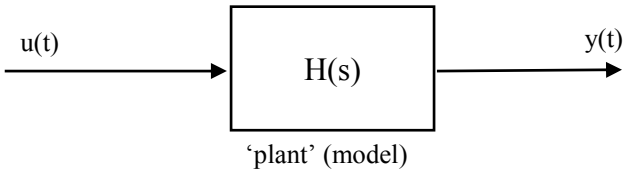
so,

$$\dot{\underline{x}} = \begin{pmatrix} 0 & 1 & 0 \\ -{}^{K}\!/_{M} & -{}^{B_s}\!/_{M} & {}^{K}\!/_{M} \\ 0 & 0 & 0 \end{pmatrix} \underline{x} + \begin{pmatrix} 0 \\ {}^{B_s}\!/_{M} \\ 1 \end{pmatrix} u$$

(A) *(B)*

Hold on ! You can't have two different sets of matrices which both describe the same model !? Yes you can – the definition of the state vector is different in each case, so the model will be different in structure, but it will have the same output. The states can be seen as 'intermediate variables' – they are a set of variables which provide a full description of the modelled dynamics, but they aren't necessarily unique.

In many situations (eg for automatic control) we consider the system as a transfer function between input(s) and output(s) :



As the states may vary depending on how you set the model up, the second equation in equation 12 is used to 'translate' the states into one or more recognisable outputs. Matrices *C* and *D* are not essential to the valid analysis or simulation of a system, but they do let you define a system from input to output. In our example, if I want the outputs to be y_1 = suspension deflection and y_2 = absolute body velocity, I need (for the two state version),

$$y_1 = x_1 , \qquad y_2 = x_2$$

so, by 'reverse population' of the matrices, it follows that :

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ (the identity matrix)}, \quad D = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

To output the same values from the three state version of the model you need (from the state definitions of that version) :

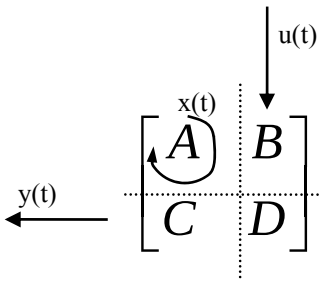
$$C = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} , \quad D = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

You can also get acceleration from this model; $\ddot{z}_b$ is also known as $\dot{x}_2$, and the equation for this is the second equation of the model. If I want to add this as a third output (to the three state model) I get :

$$C = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 1 & 0 \\ -K/M & -B_s/M & K/M \end{pmatrix} , \quad D = \begin{pmatrix} 0 \\ 0 \\ B_s/M \end{pmatrix}$$

(You can see that I've just 'imported' the second row of [A : B] into the third row of [C : D].)

Note that the matrix sizes are such that you can always put *A*, *B*, *C* and *D* together, to 'build' a rectangular matrix :



and you can think of the inputs going in, outputs coming out, and states churning around as drawn above (do I need medication ?). This has no practical use, but helps you remember how big the *D* matrix needs to be (this is often just full of zeros).

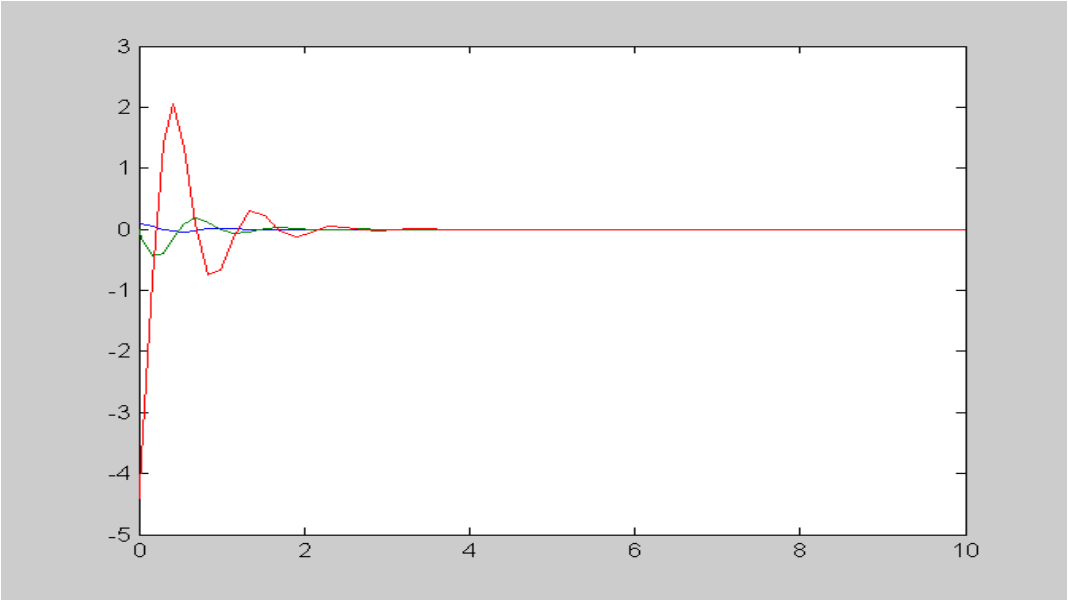
Aside : You should be able to output anything you want from a properly defined state space model (as a linear combination of the states and inputs). But, try outputting z_b from the two state version. You can't, because z_b alone does not have an independent influence on the system's dynamic response, and the number of states has been minimised to take advantage of this fact (using the combination $z_b - z_r$). This is not a problem, because you wouldn't normally be interested in variables which have no direct dynamic influence – and if you did, you could code extra states, like in the three state version.

Simulink Exercise – tutorial question

(read all of this question before starting – it will save time in the long run)

There is a state space block in the 'continuous' block library of Simulink. Create a new model which uses this block to run a simulation of the suspension model (parameters as in 'Handy Matlab at the end of Section 2b) from an initial condition of $z_b = 0.1$, $\dot{z}_b = 0$. First set up the *A*, *B*, *C*, *D* matrices and a vector for the initial conditions in a separate Matlab script file. Then you can put A, B, C etc. in the appropriate fields in the Simulink 'state space' block, and provided you've run the script file in Matlab, Simulink can 'see' the values you've defined.

Set up the model to output the body deflection, velocity and acceleration, to a 'To Workspace' block. (NB, for some *insane* reason, the 'To Workspace' blocks default their output to 'Structure' – change this to 'Array' before passing go and collecting £200.) Note that, as you've asked for a vector of outputs, and have run a simulation with values generated over time, the output gives a matrix of data (ordered in an obvious way). Include a 'Clock' (source library) in your model and send this to a time output (To workspace). Now after running the simulation you can plot the outputs against time. It should look something like this :



Why so 'jerky' ? That's because the integrator didn't *need* to take many steps to get good accuracy. It doesn't look smooth because the plot command just assumes a straight line between each point it plots. You can get a smoother looking result by specifying a 'refine factor', under 'Simulation Parameters'. Setting this to 10 will give 10 output points for each

step the integrator has to take to achieve the required accuracy. Another way, which will have benefits if you want to do any frequency domain analysis of outputs from simulation models, is to set a fixed time-step at which you want the outputs calculated. This is done by specifying a value (to replace -1) in the 'Sample time' field for each 'To Workspace' block (you need to do each separately).

References :

[1] **Thomas D. Gillespie**, '*Fundamentals of Vehicle Dynamics'*, SAE, 1992.

[2] **W.H.Press, S.A.Teukolsky, W.T. Vetterling and B.P. Flannery**, '*Numerical Recipes, The Art of Scientific Computing*', Cambridge University Press, 1992.

This page intentionally left blank. Because Microsoft Word page numbering is unreliable.

Section 3 : Ride Dynamics

These notes are an extremely condensed and reduced version of Chapter 5 from Thomas D. Gillespie's 'Fundamentals of Vehicle Dynamics'. It is well worth reading this chapter (and others) from the book – although you have to get used to American rather than SI units ! All the figures in this section come from the book.

The Ride System

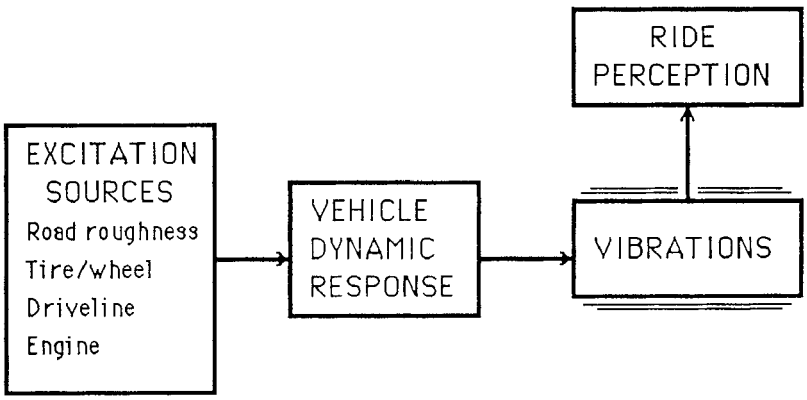


Fig. 5.1 The ride dynamic system.

This figure illustrates one of the most basic concepts in modelling and simulation, yet one which can still cause confusion. The Excitation Sources are the *inputs* to the system. These might be illustrated in the time domain, but are more often shown in the frequency domain (eg as a Power Spectral Density, PSD plot). The system (or model) then reacts to these inputs, 'converting them' into Perceived Ride in this case – which might itself be shown in terms of a measurable output, such as vehicle body vertical acceleration. Figure 5.17, later, shows an example of input PSD, system model (frequency response), and output PSD.

Excitation : Road Roughness

The principal excitation source, obviously, is the road. Fig 5.2 shows a PSD of three measured roads along with average frequency models for them. The frequency model is :

$$G_Z(\upsilon) = G_O\left[1+(\upsilon_O/\upsilon)^2\right]/(2\pi\upsilon)^2$$

where :

$G_Z(\upsilon)$ = PSD amplitude (feet^2/cycle/foot)

υ = Wavenumber (cycles/foot)

G_O = Roughness magnitude parameter ($1.25\text{x}10^5$ for rough roads, $1.25\text{x}10^6$ for smooth)

υ_O = Cutoff wavenumber (0.02 cycles/foot for rough roads, 0.05 cycles/foot for smooth)

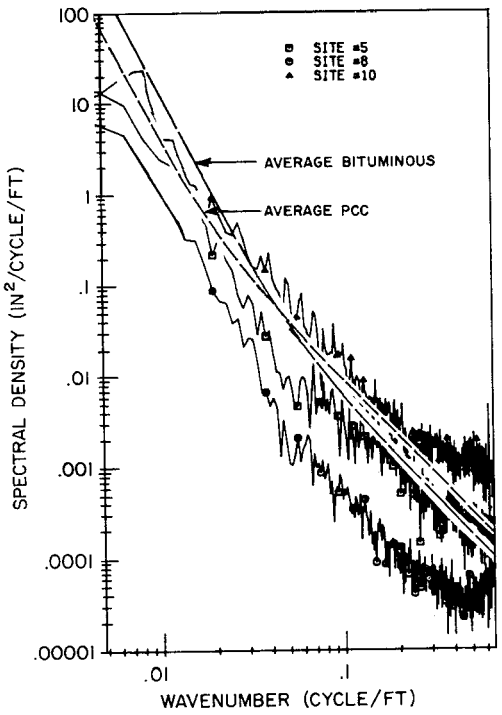


Fig. 5.2 Typical spectral densities of road elevation profiles.

The simple model can be used, along with a random number sequence, to generate test roads which are random in themselves, but yet have approximately representative frequency content. Multiplication of the original 'wavenumber' (cycles per foot) by a known (and assumed constant) vehicle speed (feet per second) gives a more recognisable PSD :

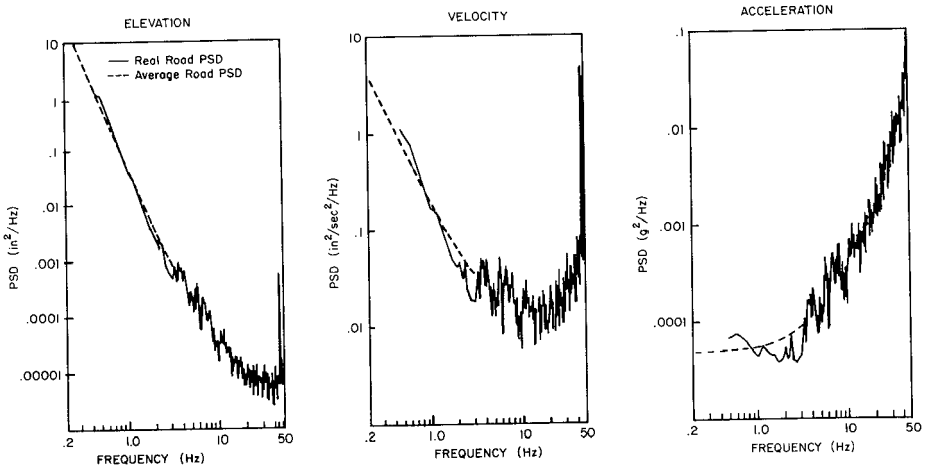


Fig. 5.3 Elevation, velocity and acceleration PSDs of road roughness input to a vehicle traveling at 50 mph on a real and average road.

Excitation : Secondary effects

Vibration from ‘imperfection’ of the wheel or driveline, and from the engine, cause further excitations, usually at frequencies higher than the principal wheel-hop and body-bounce modes that we are most interested in. Note this figure illustrating wheel runout; although the tyre is ‘complex’, the model is just a collection of simple springs – we’ll see this is quite common in basic vehicle dynamic modelling.

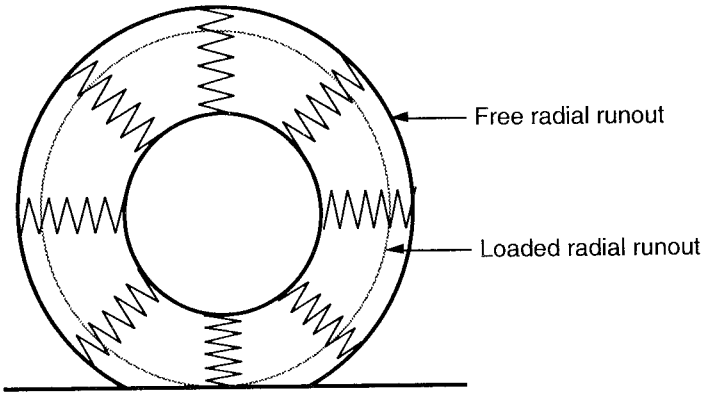


Fig. 5.5 Tire radial spring model.

‘Runout’ refers to the ‘springs’ expanding and contracting – any unevenness causes an effect similar to wheel mass imbalance, giving a cyclical input at the frequency of the wheel rotation speed. Harmonics of the ‘multiple spring’ model can also cause cyclical disturbance – did you know tyres are sometimes square ?

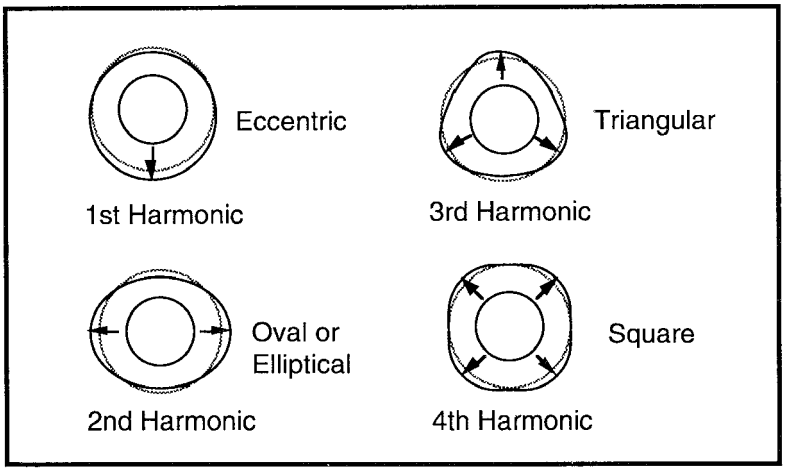


Fig. 5.7 Radial nonuniformities in a tire.

The simplest Ride model : The Quarter Car

Although simple, the quarter car model is adequate, and can be an accurate representation of real vehicle vertical motion (depending on correct setting of the parameters). This is the first model that we will build and analyse on this course.

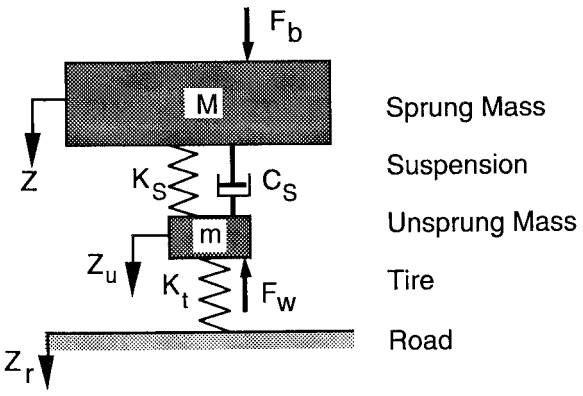
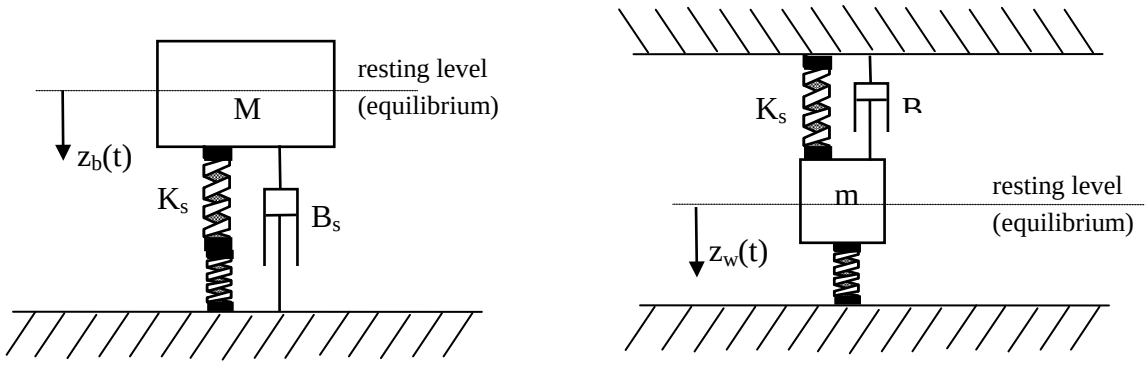


Fig. 5.14 Quarter-car model.

Typical values of parameters – large saloon :

M = 300kg
m = 30kg
K_s = 20000 N/m
K_t = 160000 N/m
C_s = 1500 Ns/m (also written as B_s below)

In order to perform some quick hand calculations on the expected frequencies and damping ratios that given parameter choices will provide, we can also think of the quarter car model in terms of *modal* (not model !) simplications. To consider body-bounce, or wheel-hop alone :



Modal simplifications of the quarter car :
(a) body-bounce only (b) wheel-hop only

You used these simple equations in part B dynamics. Firstly in the body-bounce variant, the unsprung mass is assumed zero, and the tyre and suspension springs are thus in series.

$$K_{bb} = \frac{K_s K_t}{K_s + K_t}$$

This gives a natural frequency for the body-bounce mode of :

$$\omega_n = \sqrt{K_{bb} / M} \text{ (rad/s)}$$

(divide by 2π to give natural frequency in Hz)

The 'actual' (damped natural) frequency of vibration is affected by the damping ratio, ζ

$$\omega_d = \omega_n \sqrt{1 - \zeta^2}$$

which is given according to the damping constant actually used ($B_s = C_s$)

$$\zeta = C_s / \sqrt{4 K_{bb} M}$$

Typically, a damping ratio of between 0.2 and 0.4 is achieved. What is the natural frequency, and damping ratio for the 'large saloon' figures given above ?

Looking at the 'Wheel-hop only' variant, here we assume that the sprung mass, M hardly moves at all, so the unsprung mass, m is governed by suspension and tyre springs in parallel :

$$K_{wh} = K_s + K_t$$

and : $$\omega_n = \sqrt{K_{wh} / m} \text{ (rad/s)}$$

Expected in the range 11 – 13 Hz, what is the wheel-hop frequency here ?

Ride response

Fig 5.17 illustrates, from left to right, *input*, *model* and *output*. The input and output here are the PSDs of road acceleration and vehicle acceleration respectively. (Note that it is possible to look at responses between different input/output combinations, provided the model is suitably written – eg in state space.) The model shows the gain function (frequency response, or 'bode plot' gain) between the two. Note how the road accel essentially looks similar to that given in Fig 5.3, with high magnitudes at high frequencies. These high frequencies are not transmitted through the suspension, whose principal job is to isolate the vehicle body from harsh inputs. In the model plot you can see the obvious body bounce resonance, around 1Hz, and the kink around 10Hz showing the wheel hop resonance (we will see later, when looking at eigenstructure, that *both* modes of vibration affect *both* masses, at least to some extent). The output PSD shows how well the system has attenuated the high frequencies, whilst magnifying the response around 1Hz. You can look at the plots as 'input x model = output', although you should bear in mind that, because we are viewing PSDs of input and output, which give their signal squared, the true 'maths' is 'input PSD x $(|\text{model}|)^2$ = output PSD'.

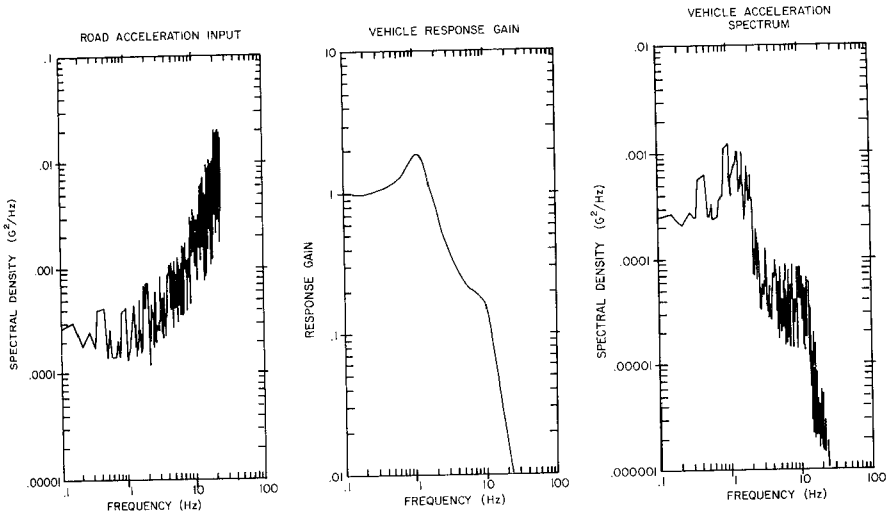


Fig. 5.17 Isolation of road acceleration by a quarter-vehicle model.

Suspension stiffness and damping

The magnitude of the body bounce peak is affected by what Gillespie refers to as the 'ride rate', K_{bb} above. Because K_s and K_t act in series, and K_t is so much stiffer than K_s, K_{bb} is dominated by K_s. In Fig 5.18 below, K_s has been tuned to give a body bounce resonance frequency varying from 1Hz to 2Hz (without varying damping factor). This shows that as you increase the resonance frequency for body bounce, so the transmission of acceleration from the road to the vehicle will increase in magnitude – by a large factor. The *y* axis here is on a linear scale, so you can look at the area under the peak as a measure of the mean-square acceleration transmitted from road to vehicle. Obviously, the aim is to achieve good isolation, and reduce transmission, so lower K_s, and hence lower resonance frequencies are aimed for.

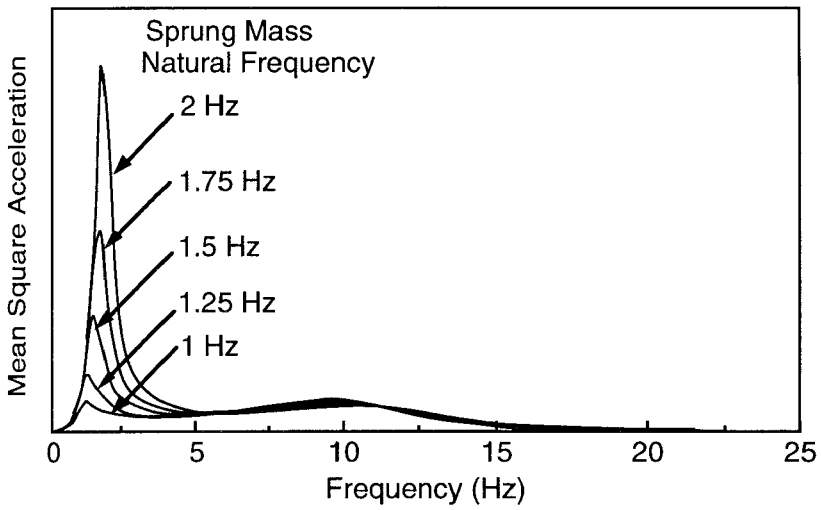


Fig. 5.18 On-road acceleration spectra with different sprung mass natural frequencies.

The factors which prevent excessively low K_s and hence lower ride rates are :

1) suspension travel – low spring rates necessitate greater suspension travel
2) handling performance is affected by fluctuating vertical load on the tyre
3) very low frequencies (well below 1Hz) would induce nausea – essentially the same as seasickness – and worsen travel sickness. In terms of practicality however, factors 1 and 2 restrict the frequency more than this factor.

Fig 5.18 isn't totally 'honest' however, as the damping in the suspension can always be increased, along with the suspension spring rate, to counter the high body bounce peak. The introduction of damping isn't a 'simple' solution though, because it reduces isolation at higher frequencies. Increases in damping essentially spread the range of frequencies through which vibrations are transmitted, whilst reducing the peak at a given (in this case body bounce) resonance :

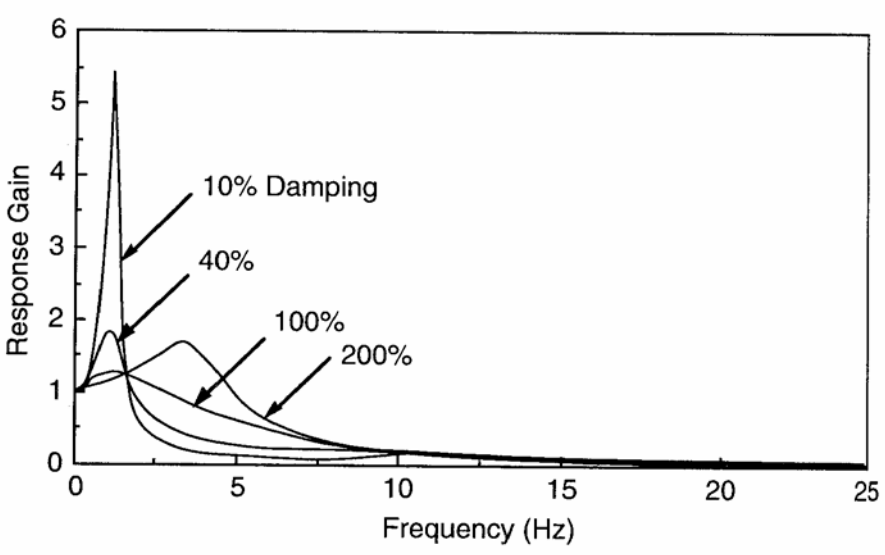


Fig. 5.19 Effect of damping on suspension isolation behavior.

Fig 5.19 shows the effect of increasing C_s to give damping ratio (ζ) in the range 0.1 to 2. Note how at the extreme (200% damping) level, the 'stiff' damper has effectively tied the body and wheel together, combining body bounce and wheel hop into one resonance at about 3.5Hz. This would be highly undesirable, due to its affect on our own body 'system'. 1Hz and 10Hz feels a lot better than 4Hz vertically – see later.

Active suspension

A little 'aside' – although it is expensive to fit and run (in terms of fuel) controllable force generators (such as hydraulics, hydro-pneumatics). They can, at least in theory, provide exceptional reductions in body bounce vibration transmission. Note that, interestingly, it is less easy to control the wheel hop peak.

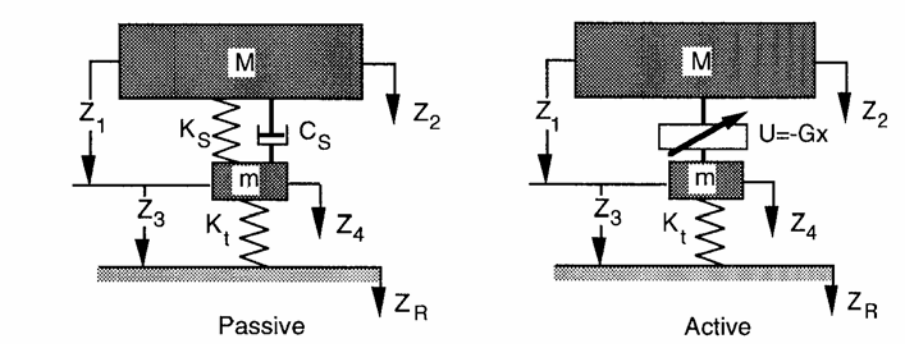


Fig. 5.22 Quarter-car model representations of passive and active suspensions.

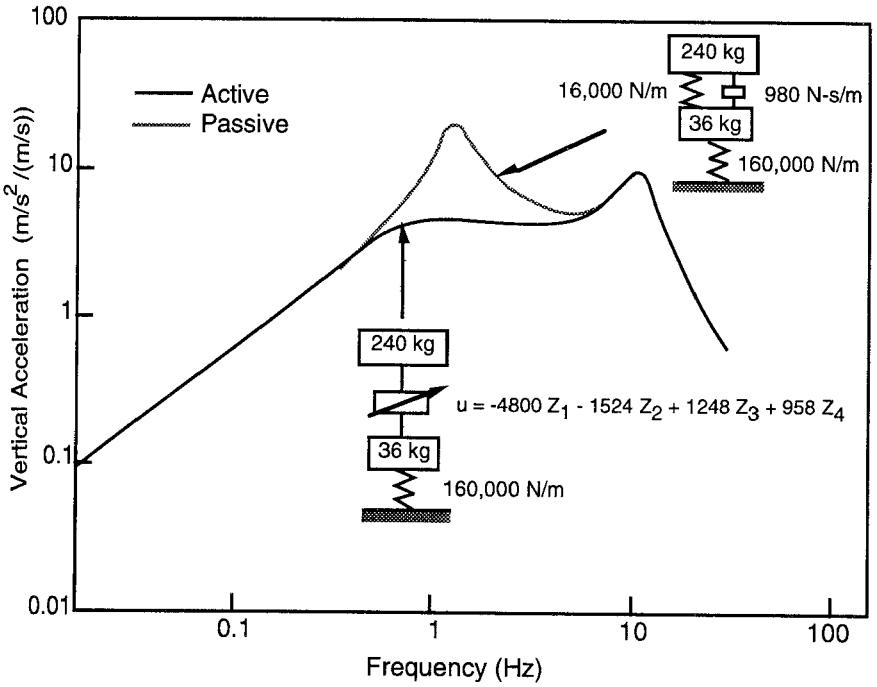


Fig. 5.25 Comparison of vertical acceleration response of active and passive systems [42].

Bounce and pitch in combination

The quarter car model is good for investigating 'bounce only' motion of the vehicle, but more complex models are needed to give a more complete insight into why suspensions are designed the way they are. The natural progression is to include pitch motion, using the 'half car' model. Note that, although the analysis and discussion of this system gets more complicated – with front and rear suspensions being tuned to complement each other to give both desirable pitch and bounce – the model is not much more complicated. Fig 5.32 shows that we need only a rigid beam suspended on two springs to get a useful simulation model. The equations for this type of system have, once again, been derived in part B dynamics.

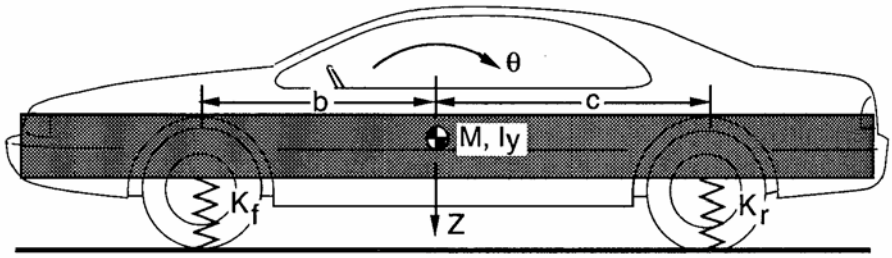


Fig. 5.32 Pitch plane model for a motor vehicle.

I won't get into the details of how the pitch and bounce modes behave, and how the ride rates of front and rear can be tuned here. Gillespie does an admirable job within a few pages in his book. It is worth noting a few basic issues though :

(a) 'Wheelbase filtering'

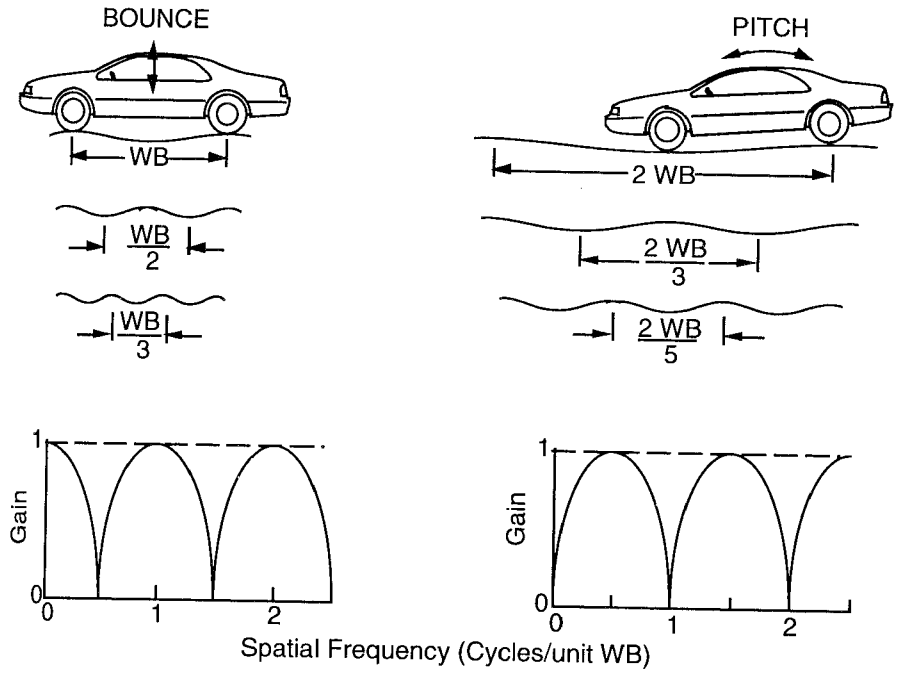


Fig. 5.29 The wheelbase filtering mechanism.

Fig 5.29 shows how the spacing of the front and rear suspensions can couple with the wavelength in the road to give 'just bounce' and 'just pitch'. The effect is not really significant in practical terms, (very few roads look like single sinusoids !) but it is an effect worth understanding when it comes to interpreting PSDs or frequency responses from more complex systems.

(b) Position and direction of measurement

What you feel depends on where you are and in what direction you are interested, aswell as on the type of motion. When we looked at the quarter car, we consider only vertical motion, but think about the simple beam of Fig 5.32 in pitch motion about its centre of gravity; if you 'sit' at the centre the rotation causes no motion. If you are above the C of G, the acceleration is fore-aft (not vertical), and only at the ends does pitch look 'mostly vertical'. It is interesting to note that the principal 'problem' associated with pitch is the fore-aft motion it produces for the passengers (not the vertical) – in spite of the much smaller 'moment arm'.

(c) Lower ride rate at the front than the rear

A neat way of reducing the discomfort of pitching (pitch being worse than bounce, generally) is to make the front ride rate lower than the rear :

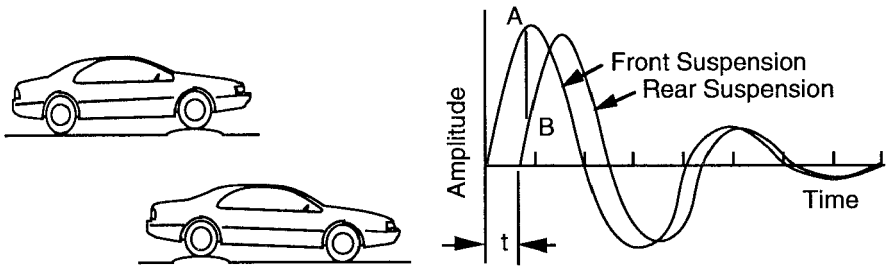


Fig. 5.35 Oscillations of a vehicle passing over a road bump.

With this design, as you meet a disturbance, you induce pitching instantaneously, but with the rear suspension motion 'faster' than the front, the motion resolves itself into bounce. Effectively, the rear oscillations 'catch up with' those at the front, eliminating the phase lag that the constant delay in the wheelbase has caused.

(d) Design 'rules of thumb' vs Modelling, simulation and analysis

Gillespie quotes one of the founders of modern vehicle dynamics, Maurice Olley, who came up with a number of design guidelines for 'good ride' from vehicle suspensions. I am absolutely confident that these constitute valuable advice which is almost certainly still used today (and you may have come across them in vehicle design modules). There is nothing wrong with using these guidelines, but you should also remember that, from Fig 5.14 (quarter car) and Fig 5.32 (half car) it is also very easy to build useful simulation models, and hence run your own simulations. From these it is possible to perform a wide range of analysis, as we will see later in this module, and hence to widen your understanding of the behaviour. You can test the guidelines and assumptions that others recommend, you can examine the influence of parameter changes and you can carry out 'what if' experiments. It is for this reason that this module concentrates on building your simulation skills rather than (only) getting you to remember useful facts and figures !

Human perception

We have looked at the PSD of ride (vertical) accelerations, and the models of suspensions which give rise to these, across frequency. In order to complete the picture about how vehicles should be designed however, we also need to consider the dynamics of the human body. Vibrations start at the road, or in the vehicle, and travel through the system of the vehicle (eg tyres, suspension) to the passenger compartment. From this point, the vibrations travel through the seat and passengers body before they are actually registered by the brain as uncomfortable. Thus we should consider the system of the body too.

One way of doing this is to subject people to vibration in an experiment, and then get them to rate how uncomfortable each vibration is. In practice, this is done by testing one frequency at a time, and building up a map of discomfort. The three plots below show lines of equal tolerance to various frequencies (x axis) applied at different amplitudes (y axis). The first two relate to vertical (ride) motion, and the last one relates to fore-aft motion (pitch, and also shuffle – see longitudinal (driveline) dynamics later in this course).

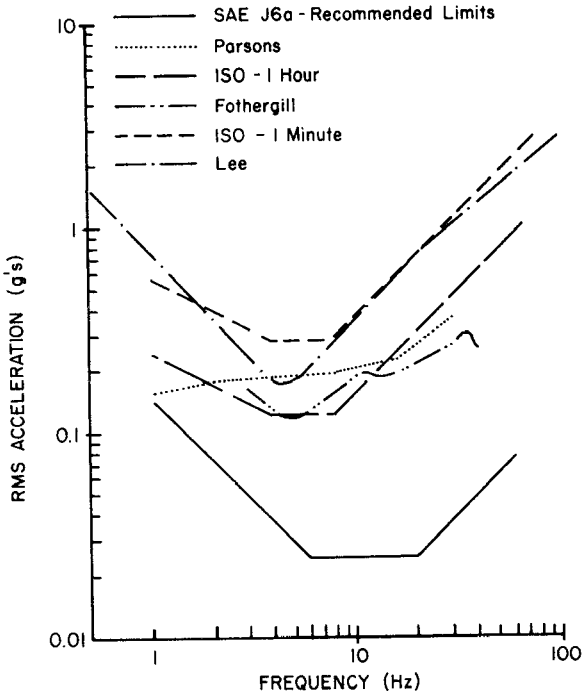


Fig. 5.36 Human tolerance limits for vertical vibration.

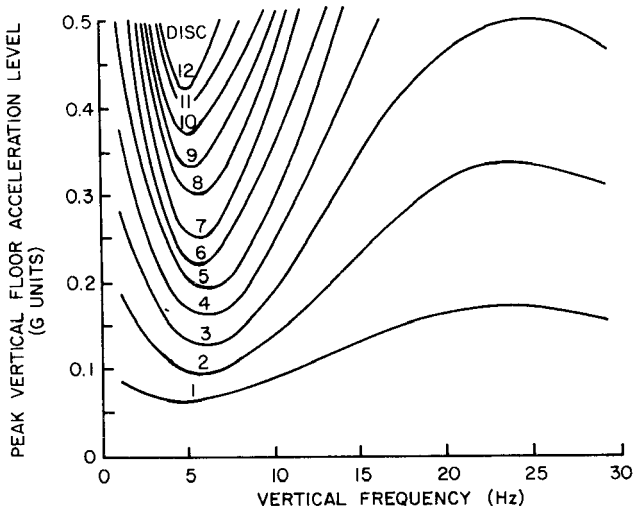


Fig. 5.37 NASA Discomfort Curves for vibration in transport vehicles.

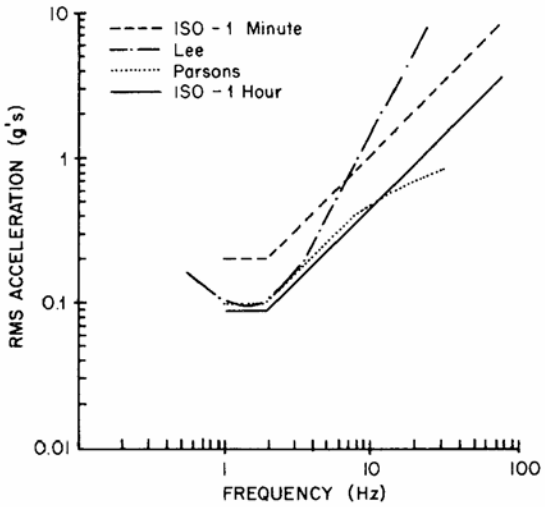


Fig. 5.38 Human tolerance limits for fore/aft vibrations.

A couple of very basic conclusions :

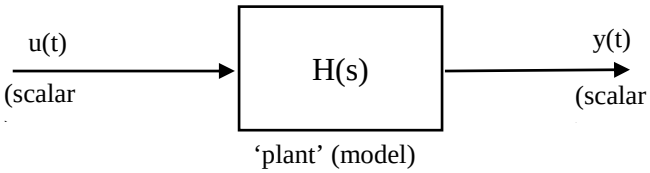
1) The NASA results correlate with *vertical* resonance frequencies of the organs of the human body in the abdominal cavity (ie your guts !) which lie in the range 4 – 8 Hz. This helps to explain why we have vehicle ride dynamics with resonance at 1Hz and 10Hz, (avoiding this range) and why the 200% damping solution in Fig 5.19 would be a disaster !
2) Fore – aft tolerance is not the same as vertical tolerance. Fore-aft vibrations cause most concern at lower frequencies (1 – 2 Hz) and less concern as the frequency rises.

Section 4 : Using Eigenvalues and Eigenvectors

Useful Properties of State Space : Eigenstructure

Right now it might seem rather a lot of work to produce models in state space form, but the advantages go further than just being able to simulate them with just one Simulink block. Matlab uses the *A,B,C,D* form as a basis for storing 'model objects', and remember that state space offers a neat way of describing systems with *multiple* inputs, to *multiple* outputs (MIMO). This has huge benefits in designing control systems; for example you can find the model for a system with many controllable inputs (eg motors in a multi-link robot arm) and many measurable outptus (eg angles between the links in the arm). A single controller can then be designed to take all the measurements, and control all the motors simultaneously, to achieve a particular goal.

In your control module, you dealt with single input single output (SISO) systems with transfer functions based on laplace transforms :



where $$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0}$$

Recall that the dynamics of such a system are determined (principally) by the solution of the 'Characteristic Equation' :

$$a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0 = 0$$

The solutions are the poles of the system, which can be plotted on real and imaginary axes, and from these, the frequency and damping of each *mode* of the system can be determined.

In the state space form, the same information is available (as the eigenvalues of the A matrix), but further information, about *mode shapes* is also available. (Don't panic, we'll review this stuff with examples.)

Consider the suspension example – I can write a SISO (laplace) transfer function, from input = road vertical velocity, to output = body velocity, as :

$$H(s) = \frac{Y(s)}{U(s)} = \frac{Bs + K}{Ms^2 + Bs + K}$$

which for the parameter choices used previously, looks like

$$H(s) = \frac{3.75s + 44.1}{s^2 + 3.75s + 44.1} \qquad (13)$$

and has solutions of its Characteristic equation, which are -1.875 ± 6.372i.

The same system in state space form (two state case) is :

$$\begin{aligned} \dot{\underline{x}} &= \begin{pmatrix} 0 & -1 \\ 44.1 & -3.75 \end{pmatrix} \underline{x} + \begin{pmatrix} 1 \\ 3.75 \end{pmatrix} u \\ y &= \begin{pmatrix} 0 & 1 \end{pmatrix} \underline{x} + (0) u \end{aligned} \tag{14}$$

Now, when you set up a state space system with a single input and single output, you can translate it into the equivalent laplace transfer function, by using the laplace property for differentiation, that :

$$\underline{|\dot{x}(t)} = s\underline{|x(t)} - x(0)$$

(where $\underline{|x(t)}$ is 'the laplace transform of *x(t)*'). This also works in vector / matrix equations, provided you follow the appropriate rules for vector / matrix algebra. Thus if we assume zero initial conditions ($x(0) = 0$) and write the laplace transforms as capitals :

$$X = \underline{|x(t)} \quad \text{etc.}$$

we get :

$$\begin{aligned} sX &= AX + BU, \qquad \rightarrow (sI - A) X = BU \\ Y &= CX + DU \end{aligned}$$

and substituting for X from the upper equation in the lower equation,

$$Y = C(sI - A)^{-1} BU + DU$$

$$H = C(sI - A)^{-1} B + D$$

This is a general solution, should you feel the need to translate a SISO state space system into laplace transfer function form. Check that it gives you equation 13 when you use the *A, B, C, D* from equation 14.

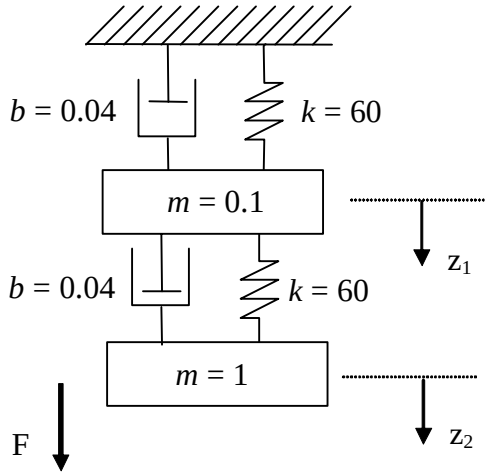
Note the terms $(sI - A)^{-1}$, which gives the denominator in the transfer function. Compare this with the definition for the eigenvalues, λ of A :

$$Av = \lambda v$$

which are found by $\det(A - \lambda I) = 0$. By inspection, you can see that the eigenvalues are the same as the roots of the Characteristic equation. Check this by using the matlab command 'eig' on matrix *A*, and comparing with the solution given above.

What is interesting here is the fact that only matrix *A* governs the fundamental modes of vibration of the system. It matters not what inputs you have, or what outputs you record, the eigenvalues give the system's full vibrational reponse – they tell you how the system will vibrate as it freely settles (with no input) after an initial condition.

It is probably easiest to show how to interpret eigenvalues (and eigenvectors, which are also useful) using an example. We need something slighly more complicated than the suspension, so we'll try two unequal masses and springs (a physical example of this will be waved about in front of you, by your friendly neighbourhood lecturer).



Start with second-order differential equations :

$$\ddot{z}_2 = F - 60(z_2 - z_1) - 0.04(\dot{z}_2 - \dot{z}_1)$$

$$0.1\ddot{z}_1 = 60(z_2 - z_1) + 0.04(\dot{z}_2 - \dot{z}_1) - 60(z_1) - 0.04(\dot{z}_1)$$

Now choose one deflection and one velocity state per mass,

$$\underline{x} = \begin{pmatrix} z_1 \\ z_2 \\ \dot{z}_1 \\ \dot{z}_2 \end{pmatrix}$$

and $u = F$.

therefore,

$$\begin{aligned} \dot{x}_1 &= x_3 \\ \dot{x}_2 &= x_4 \\ \dot{x}_3 &= -1200x_1 + 600x_2 - 0.8x_3 + 0.4x_4 \\ \dot{x}_4 &= 60x_1 - 60x_2 + 0.04x_3 - 0.04x_4 + u \end{aligned}$$

and the state space model is :

$$\underline{\dot{x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1200 & 600 & -0.8 & 0.4 \\ 60 & -60 & 0.04 & -0.04 \end{bmatrix} \underline{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u \qquad (15)$$

By setting the displacement states as 1 and 2, and velocities as 3 and 4, the first two rows of the A and B matrices have the (simple) kinematic relationships, and the last two rows have the 'dynamics'. We will see that this helps us in interpreting eigenvectors.

Modal motion in free vibration – Eigenvalues

After an initial disturbance, linear (LTI) systems will always settle, in a combination of first and second order modes, and information on these modes is very useful in helping to understand the way the system behaves. In 'Maths-speak' this means we can write $\underline{z}(t)$, the vector of deflections only, $\underline{z}(t) = [x_1, x_2]^T$ as :

$$\underline{z}(t) = \mathrm{Re}\{\underline{u}_1 e^{\lambda_1 t} + \underline{u}_2 e^{\lambda_2 t} + \ldots + \underline{u}_n e^{\lambda_n t}\} \qquad (16)$$

where n is the number of states, and each term on the right hand side represents one 'mode'. The $\underline{u}_i$ will generally be complex constant (2x1) vectors, and the λ_i are complex scalars.

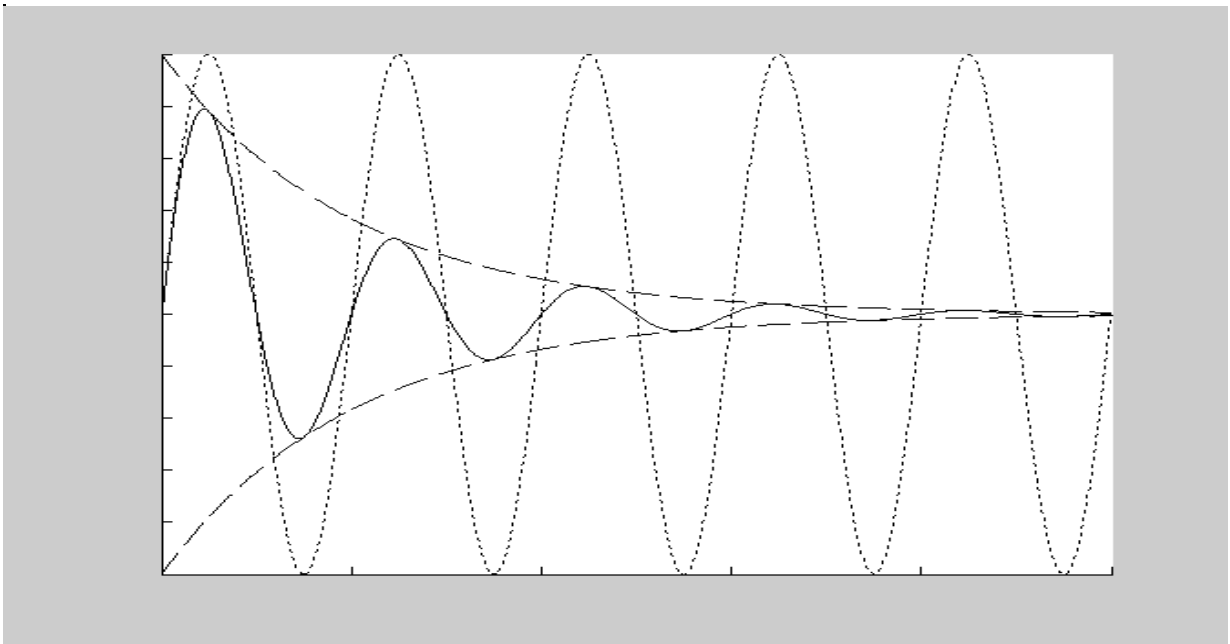
To see what each single term 'looks like', split λ into real and imaginary parts :

$$\lambda_1 = a + bi$$

$$\underline{u}_1 e^{\lambda_1 t} = \underline{u}_1 e^{(a+bi)t} = \underline{u}_1 e^{at} e^{ibt} = \underline{u}_1 e^{at}\left(\cos(bt) + i\sin(bt)\right)$$

a should be negative if your model is correct, and it relates to a physical system, because this determines the 'envelope' of the response, as a decaying exponential (shown below in dashed

lines). *b* then gives the frequency of the sinusoid component (dotted lines), (cos *bt* + isin *bt*) and the solid line shows the combination of the two.



This is then multiplied by the complex constants, $\underline{u}_i$,which affect the magnitude of the response (in this case the magnitude of z_1 compared with z_2) and the relative phase (eg 90° phase difference might have the z_1 at 0 at the same time as z_2 is at its peak).
Back to the maths :

for a single modal component

$$\underline{z}(t) = \underline{u}_i e^{\lambda_i t} \qquad \text{so if we differentiate,} \qquad \dot{\underline{z}}(t) = \lambda_i \underline{u}_i e^{\lambda_i t}$$

therefore $$\underline{x}(t) = \underline{v}_i e^{\lambda_i t}, \qquad \text{where} \quad \underline{v}_i = \begin{bmatrix} \underline{u}_i \\ \lambda_i \underline{u}_i \end{bmatrix} \tag{17}$$

Now, for free vibration response, $u = 0$, so from equation 15 ($\dot{\underline{x}} = A\underline{x}$)

$$\lambda_i \underline{v}_i e^{\lambda_i t} = A\underline{v}_i e^{\lambda_i t}$$

Therefore $$\lambda_i \underline{v}_i = A\underline{v}_i \tag{18}$$

so λ_i is an <u>eigenvalue</u> of A, and $\underline{v}_i$ is the corresponding <u>eigenvector</u>. For the spring / mass example above, the four eigenvalues are

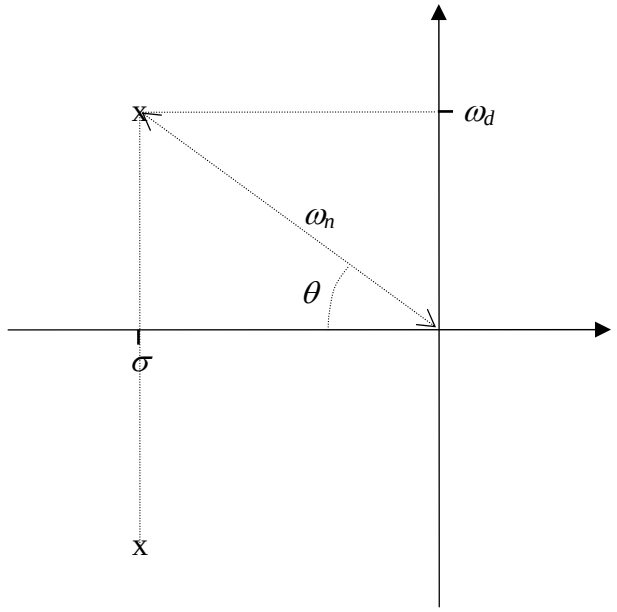
$$\left.\begin{aligned} \lambda_1 &= -0.41 + 35.08\ \mathrm{i} \\ \lambda_2 &= -0.41 - 35.08\ \mathrm{i} \end{aligned}\right] \qquad \left.\begin{aligned} \lambda_3 &= -0.01 + 5.41\ \mathrm{i} \\ \lambda_4 &= -0.01 - 5.41\ \mathrm{i} \end{aligned}\right] \tag{18a}$$

Note that the eigenvalues come in pairs, that are complex conjugates of each other, with each pair describing a mode of vibration. The eigenvalues equate to the system poles, and pole pairs can be written generally as :

$$\lambda_{1\&2} = \sigma \pm j\varpi_d$$

where σ is the modal damping factor, and ϖ_d is the damped natural frequency of the mode (in radians per second).

There are a number of useful metrics that can be taken from the eigenvalue, and some of these are illustrated in the figure below. The most commonly referred to, are the natural frequencies (I prefer to refer to damped natural frequency, because this is the frequency you actually 'see' in a physical system response) and damping ratio. Another useful metric is settling time – the time taken for the vibration to die away (to 2% of its original magnitude).



Damped natural frequency, ω_d, in radians per second.
frequency in Hz = $\omega_d/(2\pi)$

Natural frequency, ω_n, in radians per second.

Damping factor, σ

Damping ratio, $\zeta = \cos(\theta)$

2% Settling time, $T_s = \dfrac{4}{|\sigma|}$

Percentage overshoot,

P.O. = $100e^{\left(\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}\right)}$

The damping ratio varies between 0 – no damping (physically unrealistic – the system would just keep oscillating for ever) and 1 – critical damping. Note from the plot that if the damping is critical, the pole has become real (zero imaginary part), and this means the response is just given by a decaying exponential (with no sinusoid). Real poles, by definition have no complex conjugate, so they appear singly (rather than in pairs).

Classroom example :

The mass / spring and damping constants in the example above were set (at least nominally) to be similar to those of the toy shown in the lecture. Simple experiments can be done with the toy, observing its damped natural frequency, and timing the decay 'in each mode' to see how close the eigenvalues of the model are to predicting the real behaviour. (If the model parameters were tuned we could get a lot closer to reality.)

Note also, from the toy, how the modes combine, after a general 'step' or 'impulse' input to the system. Equation 16 shows that the response will generally be a combination of modes (although it is possible to isolate a single mode at a time – see eigenvectors below).

Special case : $\lambda = 0$

Given equation 16, if $\lambda = 0$, the corresponding modal component would be a constant ($e^0 = 1$). This is where the equation breaks down however – it is only valid for describing the shape of components which are *dynamically active* (ie which change over time). $\lambda = 0$ describes a 'steady state' of the system, and is best interpreted with reference to the laplace form. Because the eigenvalues are the poles, and the poles are the solution of the Characteristic equation, you can write the transfer function in terms of eigenvalues :

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{(s-\lambda_1)(s-\lambda_2)(s-\lambda_3)...(s-\lambda_n)}$$

Then you can think of *each mode* as being a 'classical first order' or 'classical second order' response, of the form :

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \qquad \text{or} \qquad H(s) = \frac{\sigma}{s+\sigma}$$

(Note how the constants used in these relate back to those used to describe pole positions in figure above.) The numerator in each case is arbitrary – it doesn't change the shape of the response in any way if it is a constant, ω_n^2 and σ just ensure that if you simulated these systems you would get unit gain in the response (output = input, after settling).

The mode for $\lambda = 0$ is a pure integrator – from the 'classical first order' form, σ =0, but with unity numerator :

$$H(s) = \frac{1}{s}$$

You'll be familiar with this, as it is the symbol used for an integrator in Simulink. In the context of a set of eigenvalues, pure integrators provide the 'freedom' in the system response for certain states to continually increase over time. It is easy to think of practical example of this – if you simulate a vehicle accelerating from rest, the states referring to vehicle velocity, and position will increase continuously – this is in contrast to the suspension model, where all the states must return to zero following a disturbance.

Modal motion in free vibration – Eigenvectors

The eigenvalues tell us what the shape of each mode looks like – what (if any) frequency, and how rapidly it decays. Eigenvectors tells us how the states vibrate *in relation to each other*. For example, in a particular mode, the eigenvalue might tell you how all the states vibrate at 5Hz, and decay in 3 seconds, but it is the eigenvector that tells you (for example) that the first state oscillates with 20 times the magnitude as state 2, and with a phase difference of 45 degrees.

Referring back to equations 17 and 18, we can write all the eigenvalues and vectors together, in a matrix form :

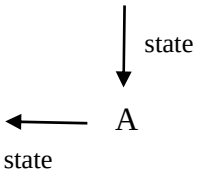
$$AV = VD \tag{19}$$

where D is a diagonal matrix of eigenvalues : $D = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_n \end{bmatrix}$

and V is the matrix of eigenvectors : $V = \begin{bmatrix} \underline{u}_1 & \underline{u}_2 & \underline{u}_3 & \cdots & \underline{u}_n \\ \lambda_1\underline{u}_1 & \lambda_2\underline{u}_2 & \lambda_3\underline{u}_3 & \cdots & \lambda_n\underline{u}_n \end{bmatrix}$

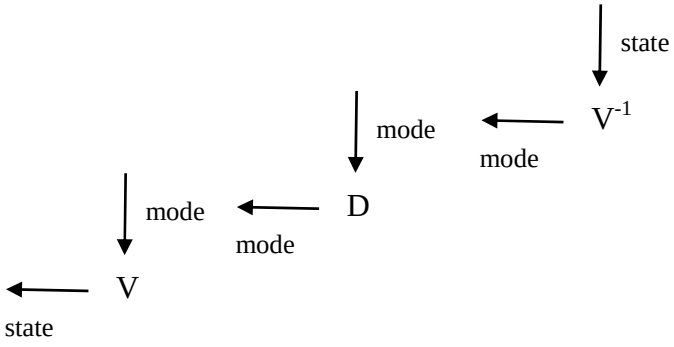
(V is just a matrix made up with each eigenvector written as a column, but I'm keeping the 'special' form that each column is $\underline{u}$ above $\lambda\underline{u}$, which arises as a result of our choice of states, and their order – displacements first, followed by velocities – this will help with interpretation of eigenvectors).

The A matrix transfers state information to state (derivative) information. Therefore the 'currency' of the input and output is the same. Graphically, you can think of this as :



which makes sense when you think of the matrix multiplication operation, whereby the inputs are laid across the top of the matrix, and the outputs come out the side (or is it just me that thinks of matrix multiplication like that ?) .

Writing equation 19 as $A = VDV^{-1}$, we can see that the V matrix acts as a 'currency transformation', translating between states and modes (because the D matrix isolates the modes diagonally, its input and output 'currency' must be 'mode'). Pictorially :



Thus when you look at the V matrix, the columns refers to the modes, in order, and the rows refer to the states, in order.

The **eig** command in Matlab, if called with the syntax **[V,D] = eig(A);** returns outputs V and D in the format described above. For the system of equation 15,

```
V =
  -0.0284 - 0.0017i   -0.0284 + 0.0017i    0.0665 + 0.0495i    0.0665 - 0.0495i
   0.0015 + 0.0001i    0.0015 - 0.0001i    0.1298 + 0.0966i    0.1298 - 0.0966i
   0.0700 - 0.9958i    0.0700 + 0.9958i   -0.2684 + 0.3593i   -0.2684 - 0.3593i
  -0.0036 + 0.0510i   -0.0036 - 0.0510i   -0.5236 + 0.7011i   -0.5236 - 0.7011i
```

Firstly, note that the second column is the complex conjugate of the first – this is because the first two columns relate to the first two eigenvalues, and these are a complex conjugate pair (equation 18a). The third and fourth columns are also complex conjugates of each other. Next, check with equation 17 (& 18a), and you find that the third and fourth rows of *V* are indeed the first and second rows multiplied by their respective eigenvalues. These two factors mean that the 'unique' information for each mode can be isolated from just ¼ of the elements of the matrix (and if you take the 'top left' data, this is as I have 'boxed' above).

Now yet more simplification can be done, because these numbers can't be used to tell us about 'absolute' magnitudes of motion; the oscillation magnitude depends on how large the input is. The easiest way to interpret each eigenvector is to divide all the (in this case two) elements by the element with the largest magnitude. For the first mode then, dividing the two values in the box on the left by `-0.0284 - 0.0017i`, we get :

```
   1

-0.0512 - 0.0000i
```

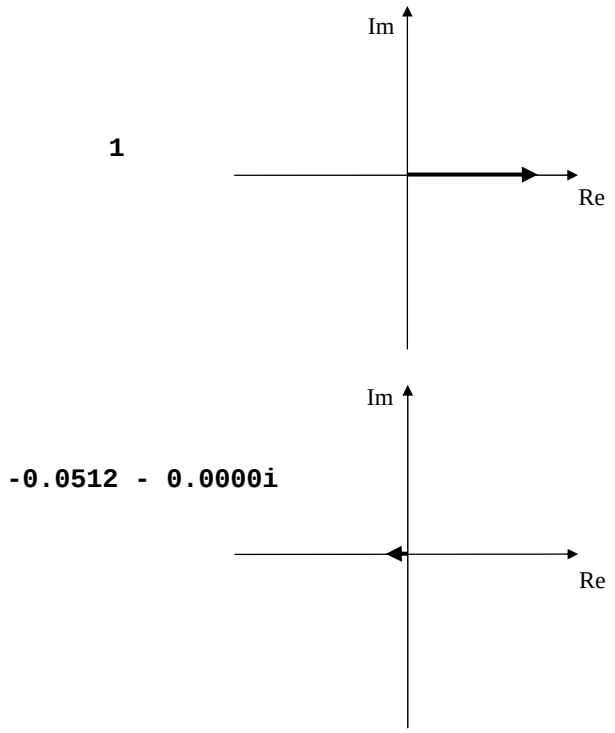
Aside : Division of complex numbers :

In Matlab, just use divide, but make sure the 'whole' complex number is included, eg **`(-0.0284 - 0.0017i)/(0.0015 + 0.0001i)`** not **`-0.0284 - 0.0017i / 0.0015 + 0.0001i`** While we're on the subject, note that when using complex numbers in Matlab, the transpose operator, ' actually takes the *complex conjugate* transpose. If you want to take the transpose of a matrix or vector without converting to complex conjugates, you need to use .' Eg try **`[1+1i, 2+2i]'`** and **`[1+1i, 2+2i].'`**.

If Matlab is not available, the technique by hand goes something like :

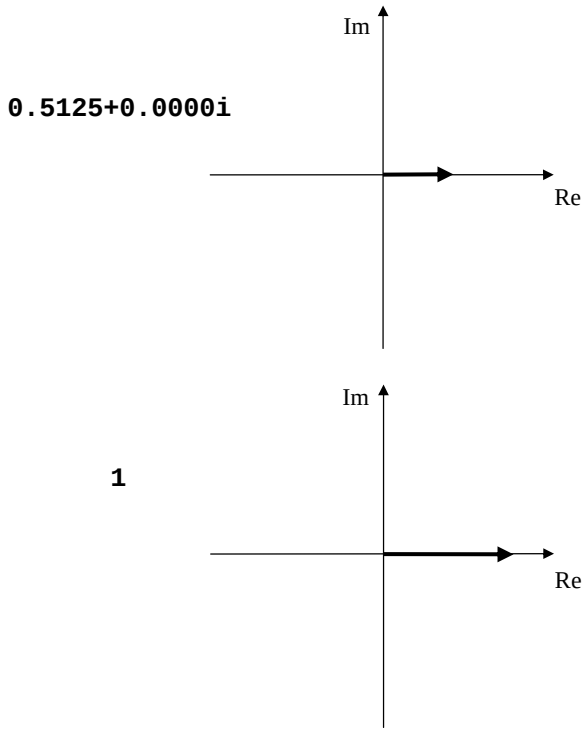
$$\frac{a+bi}{c+di} = \frac{(a+bi)(c-di)}{(c+di)(c-di)} = \frac{ac+bd+(bc-ad)i}{c^2+d^2}$$

Now if I draw the (now normalised) eigenvector components on real and imaginary axes,



What I've generated is a 'picture' of the relative magnitude and phase of the displacement states for this two mass system, when it is oscillating in the first mode. You can visualise the motion by imagining both vectors rotating, from this position, at constant angular velocity – the translation you would see in the masses is given by the projection of each vector onto either the real or imaginary axis (it doesn't matter which, but as the motion is vertical it is probably easiest to visualise this case in the imaginary axis). For this first mode, the upper mass oscillates, and the lower mass moves only about $1/20^{th}$ of the distance covered by the upper mass. The lower mass is also almost 180° out of phase (it moves up when the upper mass moves down, and vice versa).
This mode is easily demonstrated on the classroom 'toy'; from the eigenvalue we can see it is the 'faster' mode, which settles relatively quickly. Using the metrics above, the frequency is 5.6Hz, and the settling time is about 10 seconds.

The other mode (from the third and fourth eigenvectors) looks like this :



In this case the masses move in phase with each other, with the lower mass covering about twice the deflection of the upper mass. This is the 'slower' mode, with the eigenvalues showing a frequency of about 0.9Hz, and a very long settling time of about 400 seconds (7 minutes !).

Eigenstructure : Concluding remarks

The eigenstructure provides a useful, quick means of a) predicting what you would expect a physical system to do, from its model, and b) interpreting the responses you experience in a physical system. There is also value in that you don't need to run simulations to get the information – it is available directly from the *A* matrix.

The eigenvalues can also be useful when you're building a model – if you've got the direction of forces muddled up in a simulink model, the outputs may become unstable, and you can check this by checking the eigenvalues (positive real parts indicate unstable (and probably wrong) poles).

Eigenvectors can be useful in the diagnosis of vibration problems – if one component of a physical system is failing in service, there may be a mode which causes it to vibrate disproportionately.

Another use of eigenvectors is that they can be used to isolate an individual mode; if you run a simulation from an initial condition, where the initial condition is 'equal' to the eigenvector, the corresponding free motion will only include that mode. Of course, the eigenvector is complex, so in practice you set the initial condition according to the real, or imaginary, part of the eigenvector. This is the same principle as imagining the vector rotating, and projected onto the real or imaginary axis – what you are doing is performing a 'freeze frame' on the system as if it were oscillating in a particular mode – the free motion from this point on, will settle with all states following the given mode.

Try this with the model of equation 15 (you'll need to specify suitable output matrices, *C* and *D*). Try running a simulation from an initial condition in each of the two modes, and then see what happens when you choose a more general initial condition, say `>>x0 = [1 1 1 1]';`

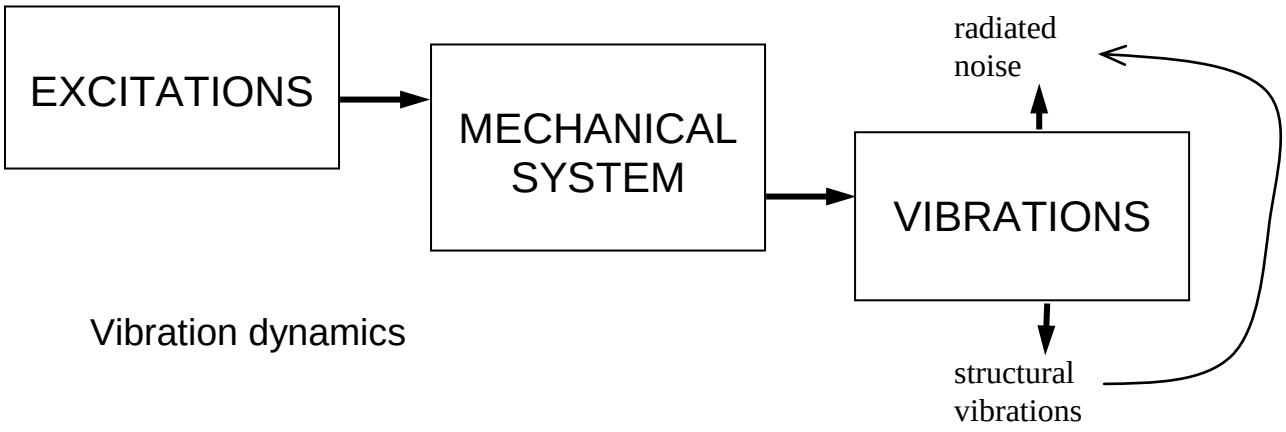
Section 5 : Drivetrain Dynamics

This part of the course deals with the longitudinal behaviour of the vehicle – how the engine torque is converted to longitudinal motion, and how the torsion that is generated in the drivetrain components induces (perhaps unexpectedly) oscillatory dynamics which affect the way the driver relates to the vehicle – influencing 'performance feel'.

Contents :

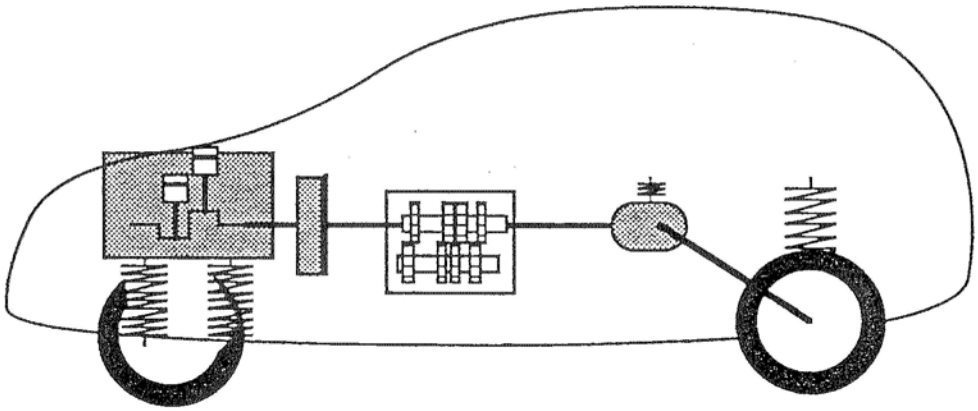
1 Overview

Mechanical vibrations can be analysed in terms of INPUT, SYSTEM and OUTPUT :



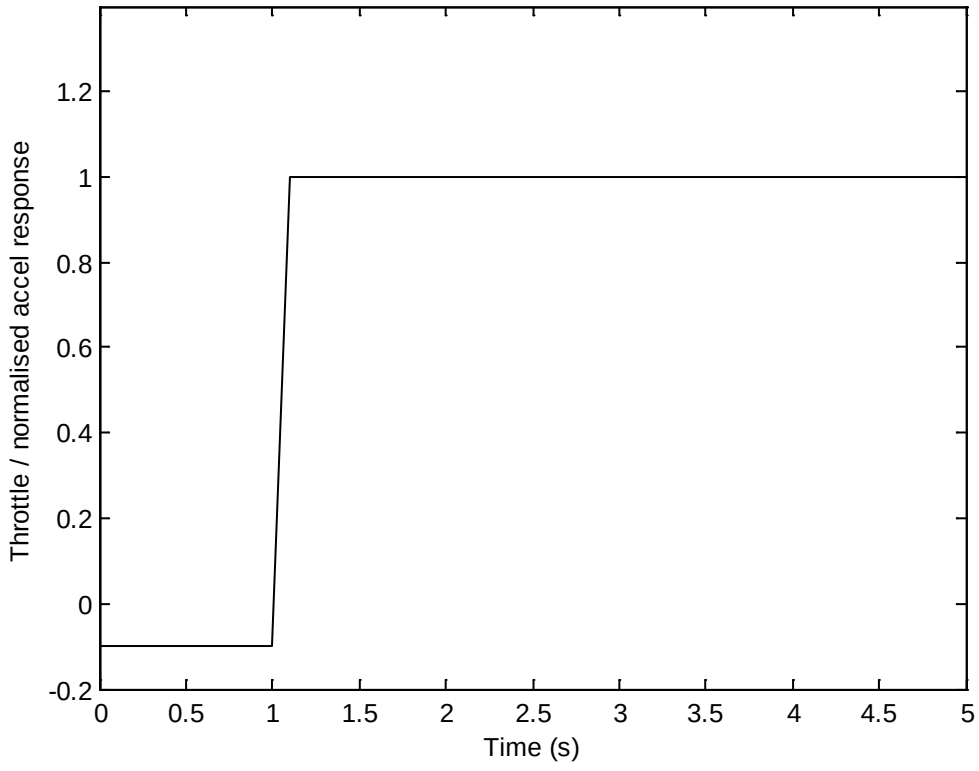
In this case, all three play an important role (not just the SYSTEM).

The drivetrain system includes the engine, clutch, transmission, driveshafts, wheels, tyres and ultimately vehicle body. Engine mounts and suspension also have a (secondary) effect :



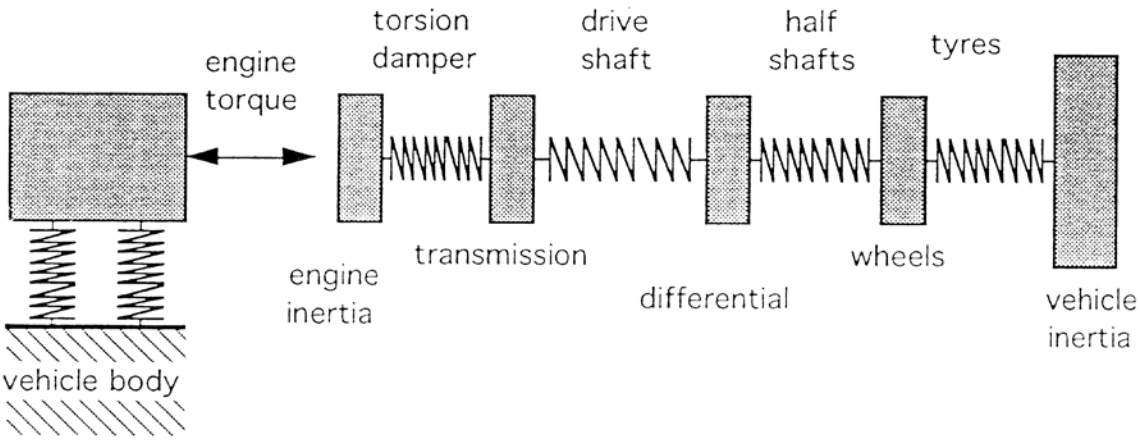
Isn't the transmission of force to the body essentially 'stiff' ? No – high forces and compliance mean that various parts of the rotating system can be twisted, leading to torsional vibrations.

Under an aggressive throttle input (eg a step), what acceleration response will the vehicle have ?



The Drivetrain as a Vibrating System

Mathematical models are usually based on simplified engineering concepts. For the drivetrain, the various components can be represented as rotating masses and torsional springs. The drivetrain is then mounted to the vehicle body at various locations (especially engine mounts and vehicle suspension) providing structural paths for vibration transmission.

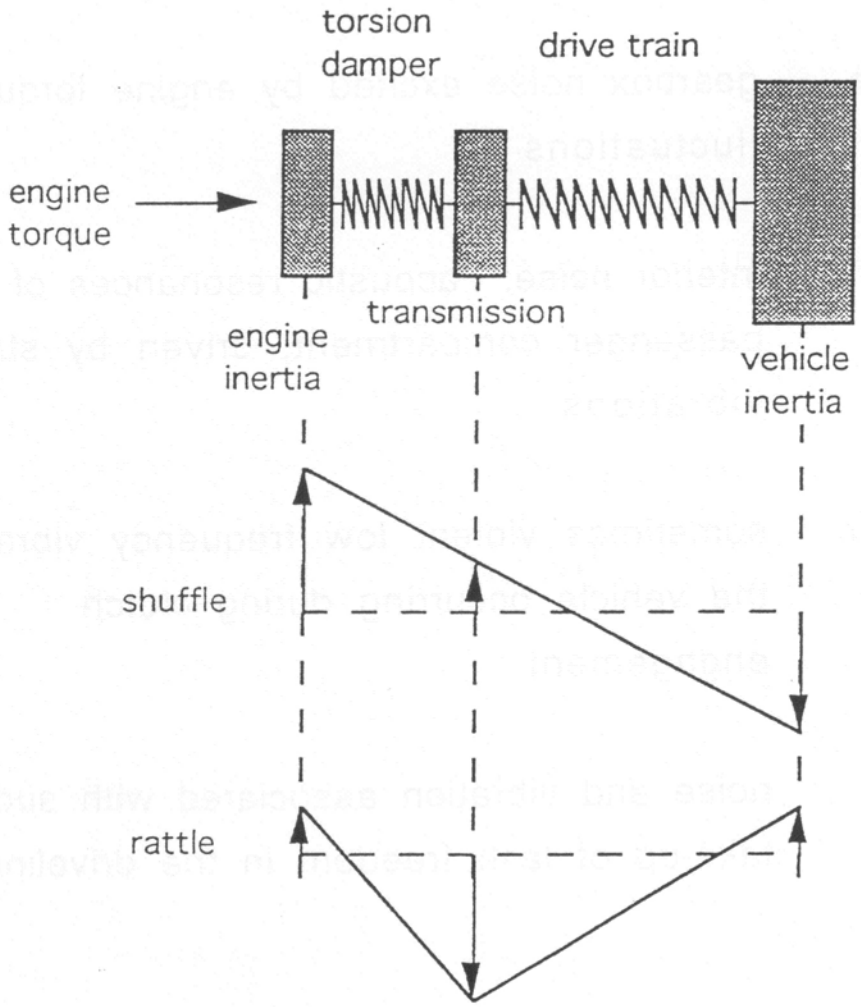


Control can be achieved by isolating or reducing the vibration. This can be tackled by :

- Modifying the system dynamics (the MECHANICAL SYSTEM) – eg by adding damping or shifting resonance frequencies
- Modifying the inputs (EXCITATIONS), to reduce their amplitude or shift their component frequencies

The Principal Degrees of Freedom

The simplest useful model of the drivetrain has three degrees of freedom :



There is also a third mode – simply rigid-body rotation of the entire drivetrain.

Each mode has its own natural frequency : typically 2 – 5 Hz for ‘shuffle’ (or surge) and 40 – 80 Hz for ‘rattle’.

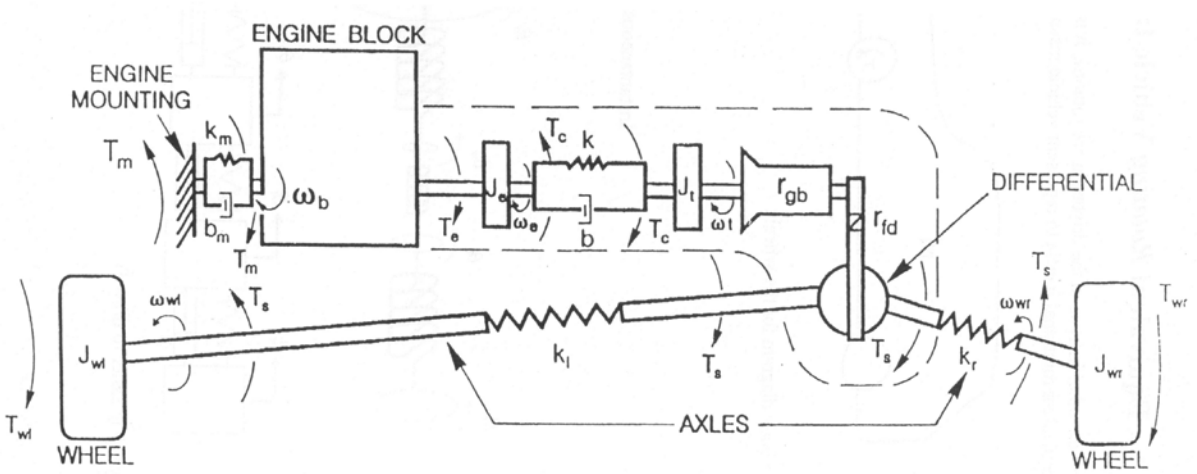
Main Symptoms of Driveline Vibration Problems

- **shuffle** uneven acceleration of the vehicle, particularly after 'tip-in'
- **rattle** gearbox noise excited by engine torque fluctuations
- **boom** interior noise : acoustic resonances of the passenger compartment, driven by structural vibrations
- **judder** sometimes violent low frequency vibration of the vehicle occurring during clutch engagement
- **clunk** noise and vibration associated with sudden take-up of lash freedom in the driveline

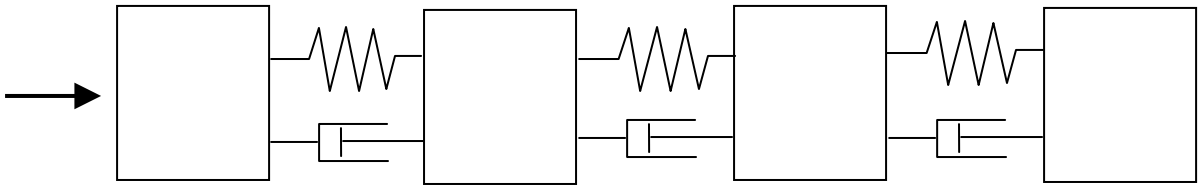
Which of these are examples of free vibration, and which are forced ?

2 A Basic Torsion Model for the Driveline

Driveline Model Schematic for a Front Wheel Drive Vehicle

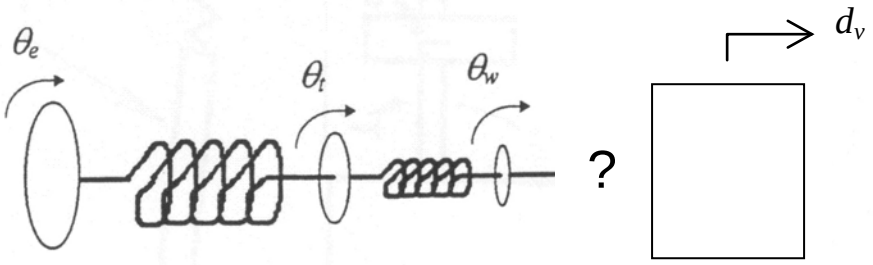


Representing a FWD Vehicle with a Four Mass Structure



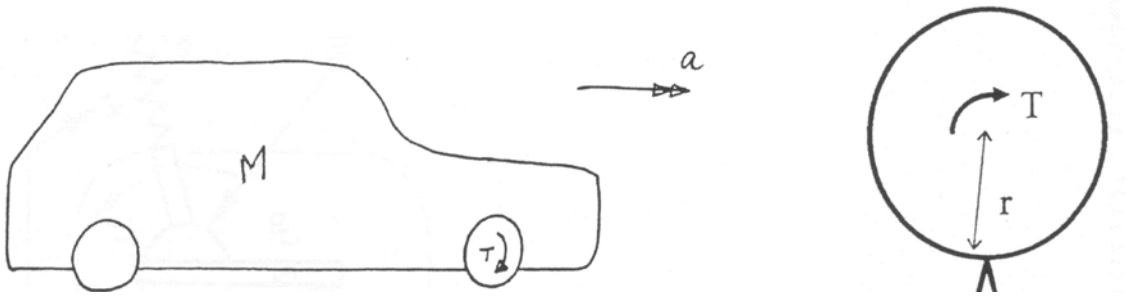
Rotation or Translation ?

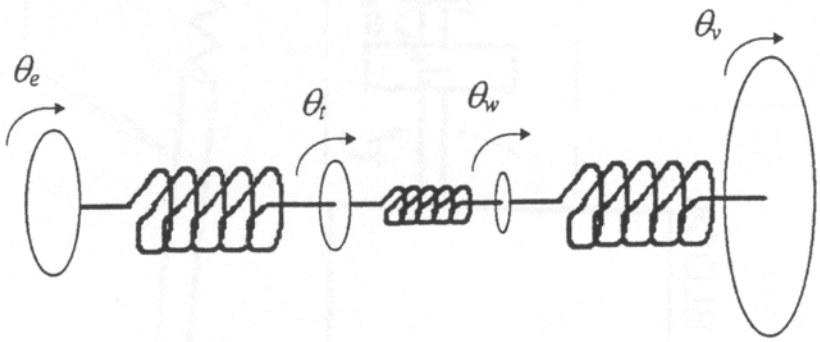
Most of the inertia components in the drivetrain are rotational, so are described in terms of their moment of inertia, but the tyre translates rotation to translation in the vehicle body :



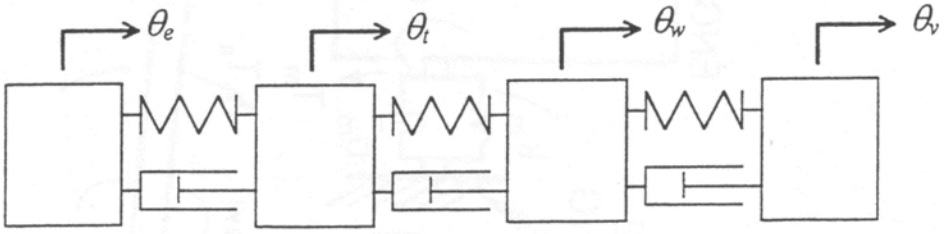
Equivalent Rotating Vehicle Inertia

It's easier to visualise the system as if it is translational (four mass structure above), but mathematically, the model is easier to formulate entirely as rotational (we'll see this for the tyre, later). Consider the vehicle body and wheel components as separate free body diagrams :



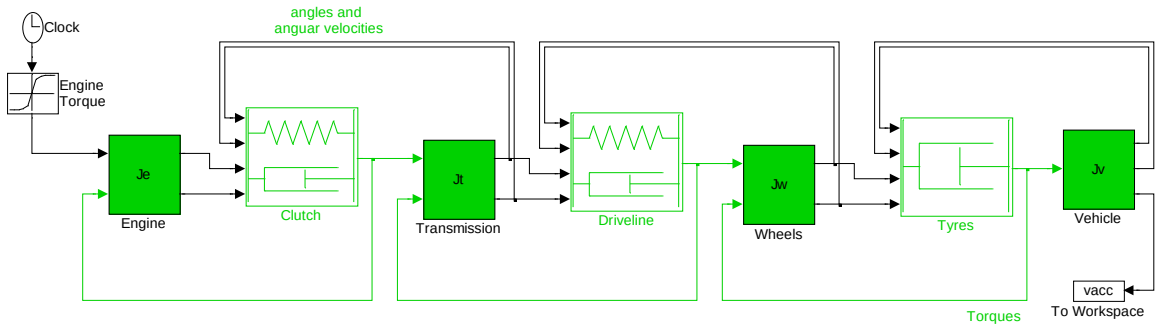


or,



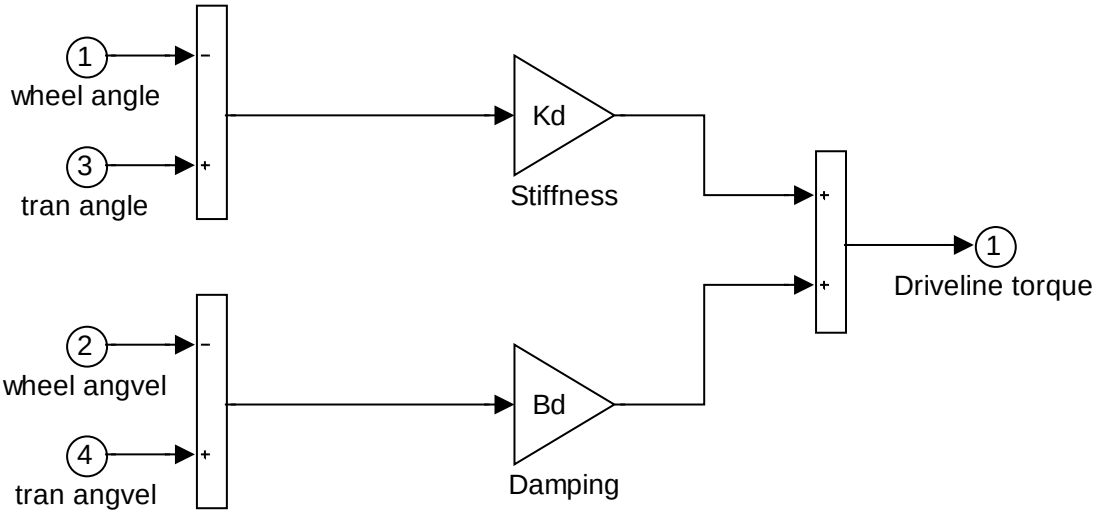
(which is easier to draw !)

The Four Mass Model in SIMULINK

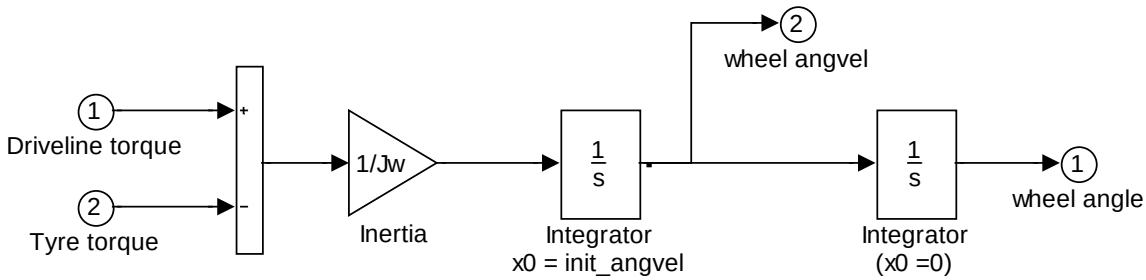


'Standard' Model Components

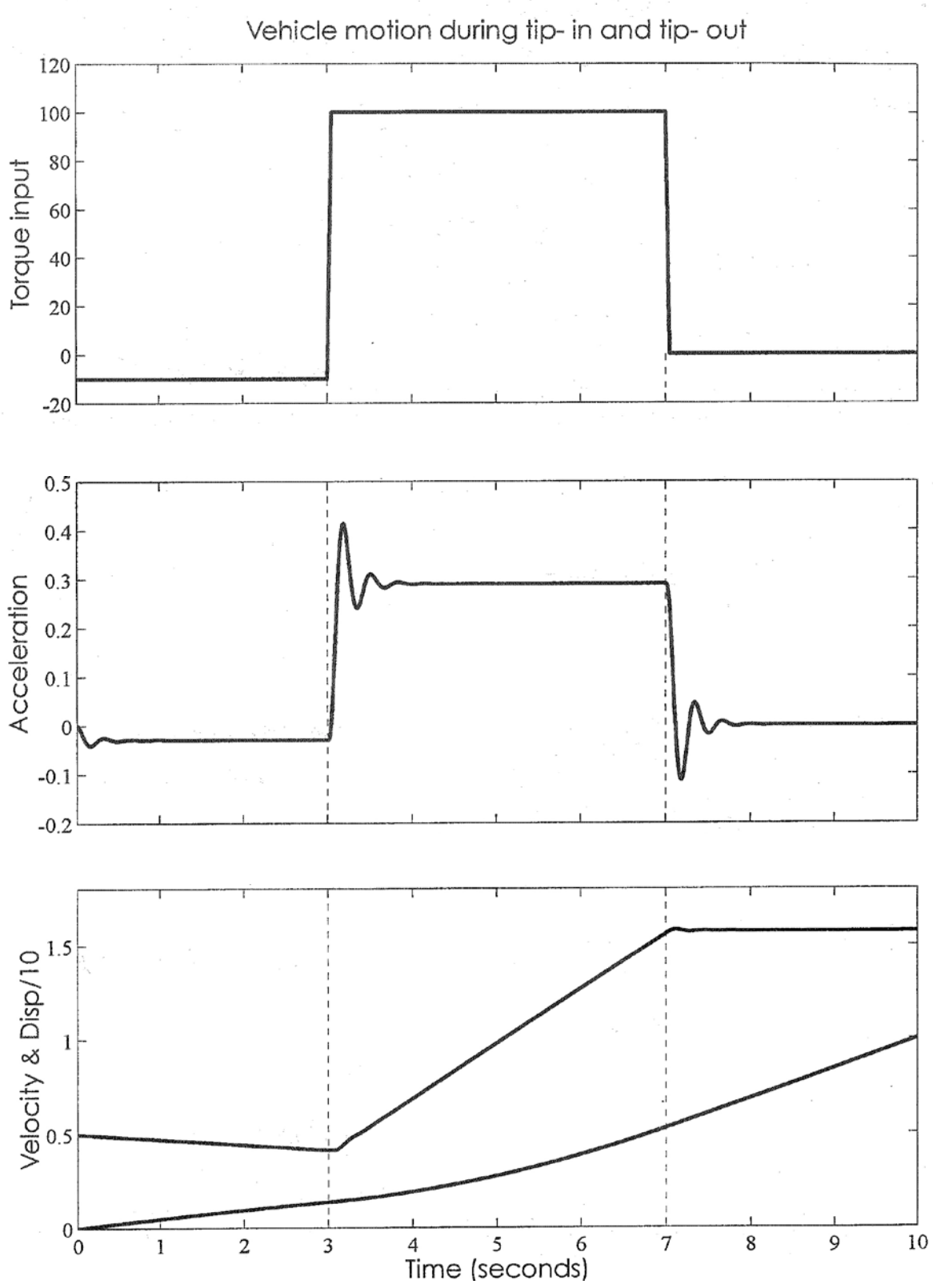
Spring Damper component (calculates 'forces' to apply to masses, from relative motions)



'Mass' component (calculates motion on a single mass, from 'forces' (torques) applied)



Basic (linear) simulation response :



3 Sources of Excitation

Why do clutches have springs built into them ?

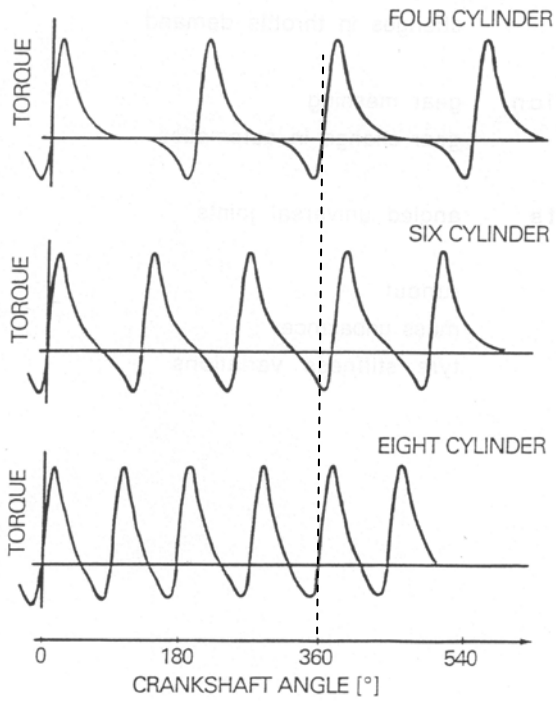
Principal Excitation Sources

- **engine** combustion
  irregular combustion
  changes in throttle demand
- **transmission** gear meshing
  gear changes (in automatics)
- **driveshafts** angled universal joints
- **tyres and wheels** runout
  mass unbalance
  tyre stiffness variations

Which of these are forcing *excitation, and which result in a* free *vibrational response ?*

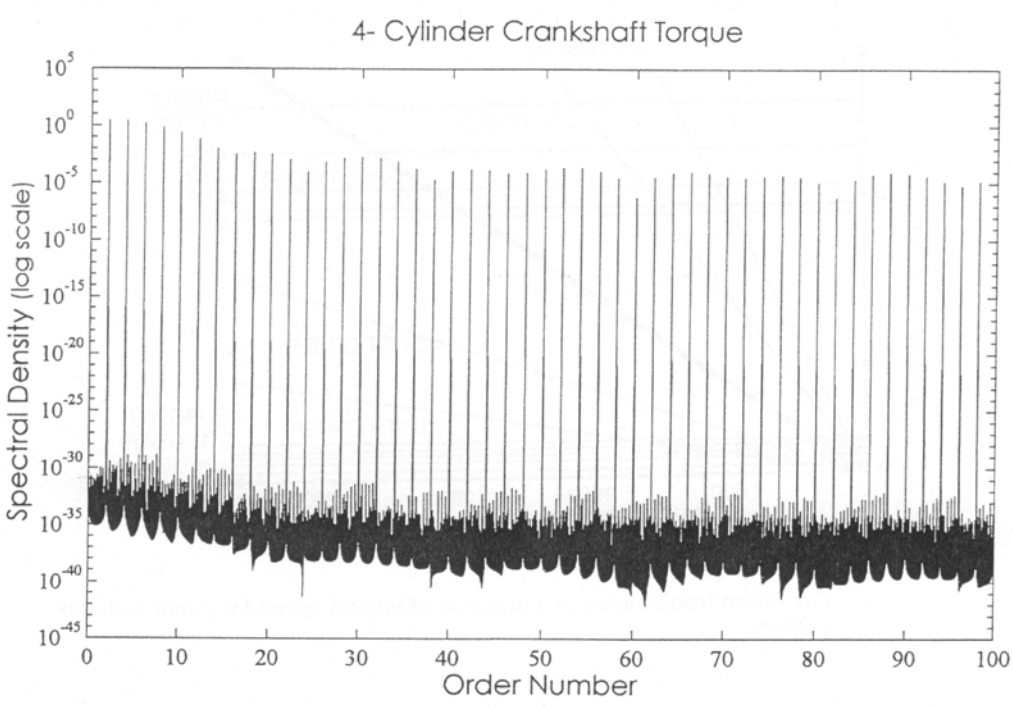
Excitation from the Engine Combustion cycle

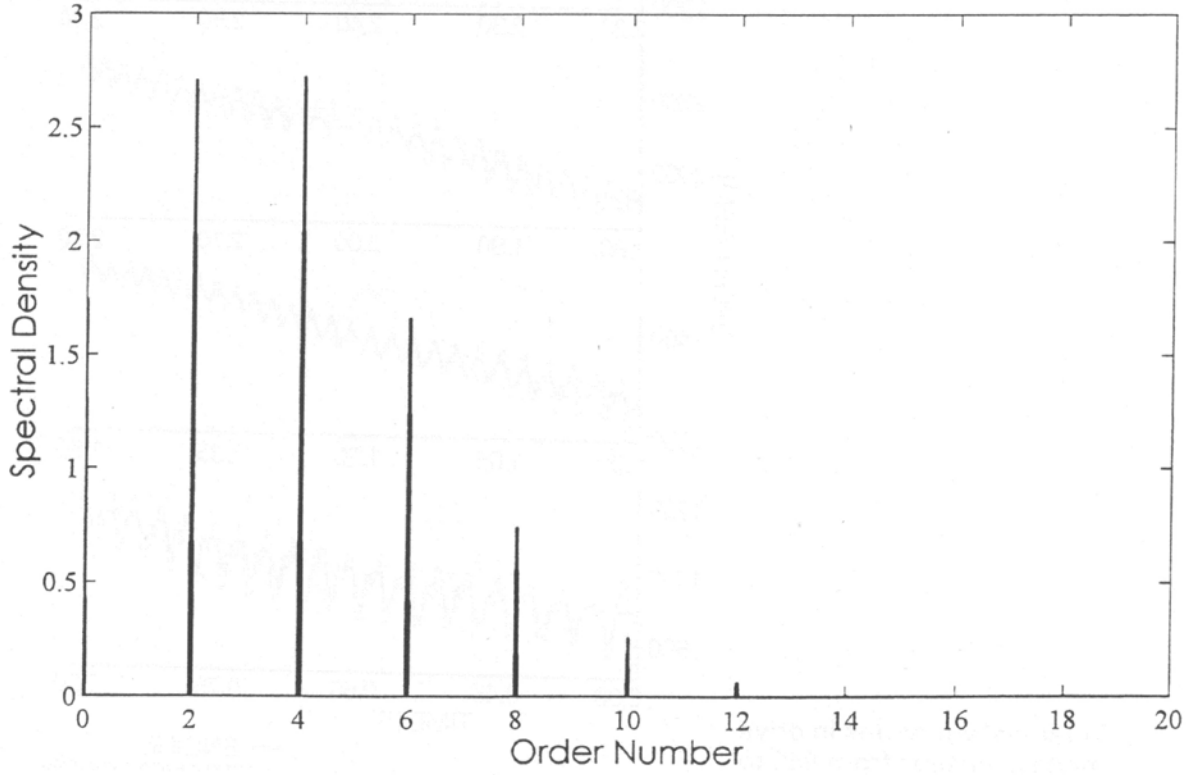
The total torque arising from all cylinders is oscillatory, with magnitude depending on where each cylinder is in the suck / push / bang / blow cycle.



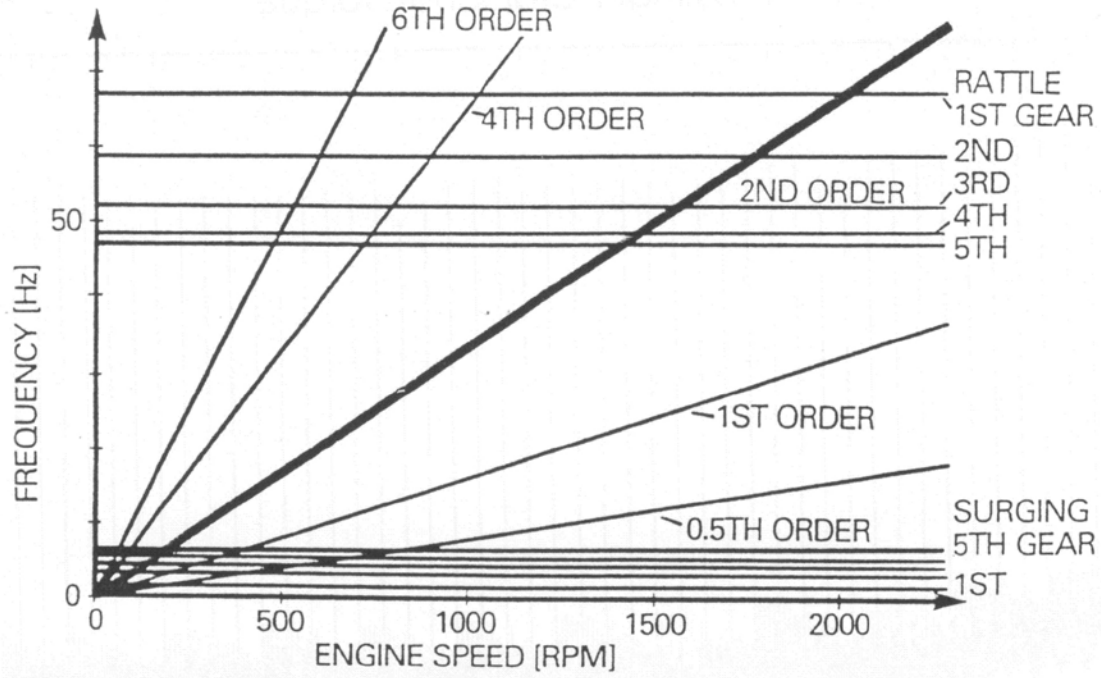
(NB six cylinder response isn't properly aligned with 360 degree angle)

After sampling the four cylinder 'time history' an FFT is used to find its frequency content, shown below as 'order number' (ie normalised against the engine rotation frequency).





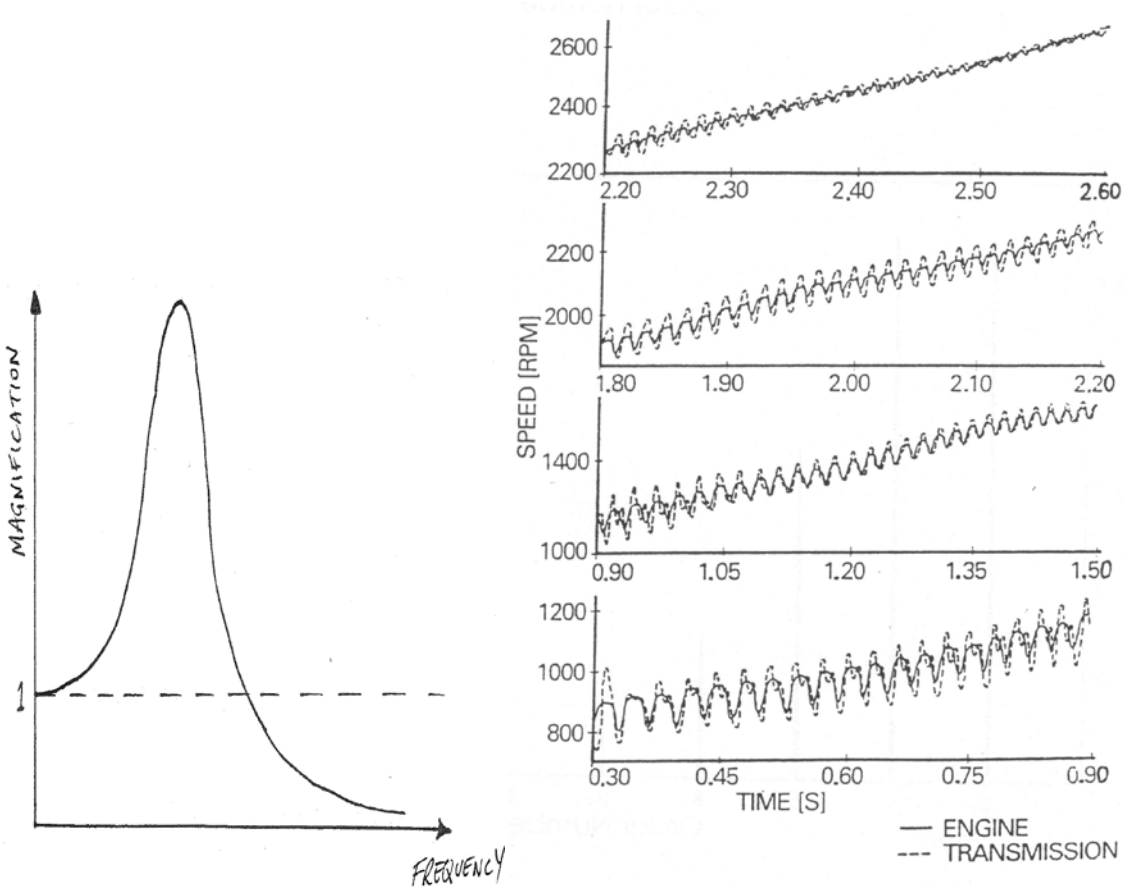
Consider how each order of this combustion signal will change with engine speed, and when each order will coincide with one of the system's resonance frequencies (fixed for each gear).



Excitation frequencies as a function of engine speed for a four cylinder engine

NB : The 1st order response here just acts as a 1:1 mapping from engine speed units (RPM) to frequency (Hz). Eg 1000 revs per minute = 1000/60 = 16.67 revs (cycles) per second (Hz).

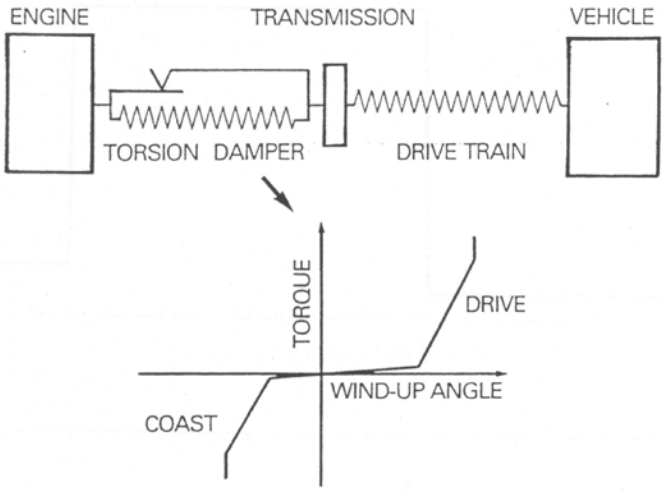
The vibration gets transmitted through the driveline when each order passes through the relevant (rattle) resonance frequency. Note that damping within the system 'spreads' the effect across frequencies, with peak vibration transmission at the 'crossover' point.



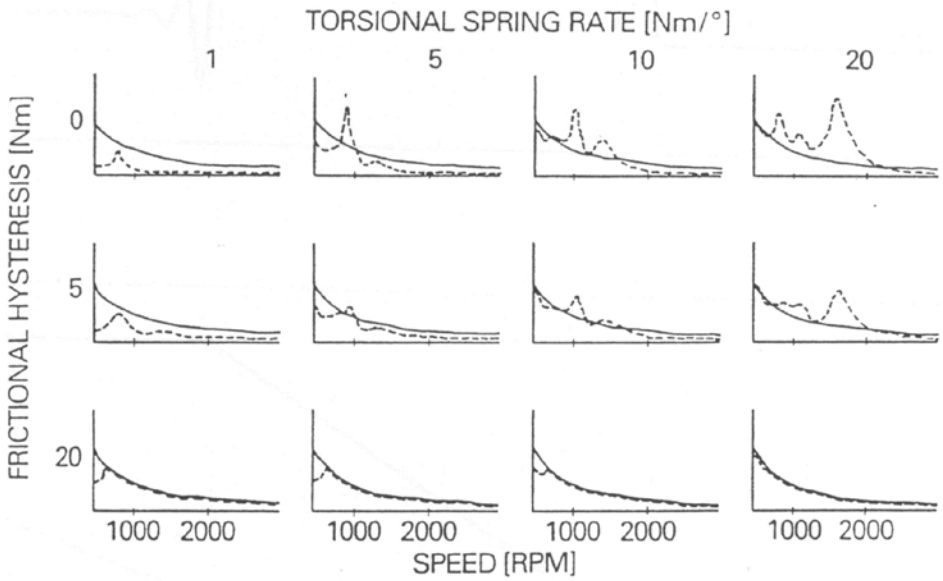
4 A Closer look at Driveline Model Components

The clutch springs (torsion damper)

The clutch isolation springs are the 'first defence' against the transmission of vibration from the engine combustion cycle. They also provide damping, through friction with their housing. A combination of different rated springs is used :



Torsion damper characteristic curve in analytical model



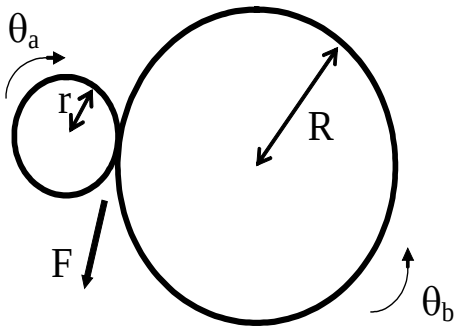
Effect of torsional spring rate and frictional hysteresis on the resonance curve of a four-cylinder passenger car engine

Which of these 12 settings is the best ? Why not apply this setting alone ?

The gearbox

A sequence of gears can be modelled as a single pair :

T_a = Torque input to *a*



T_b = Torque output from *b*

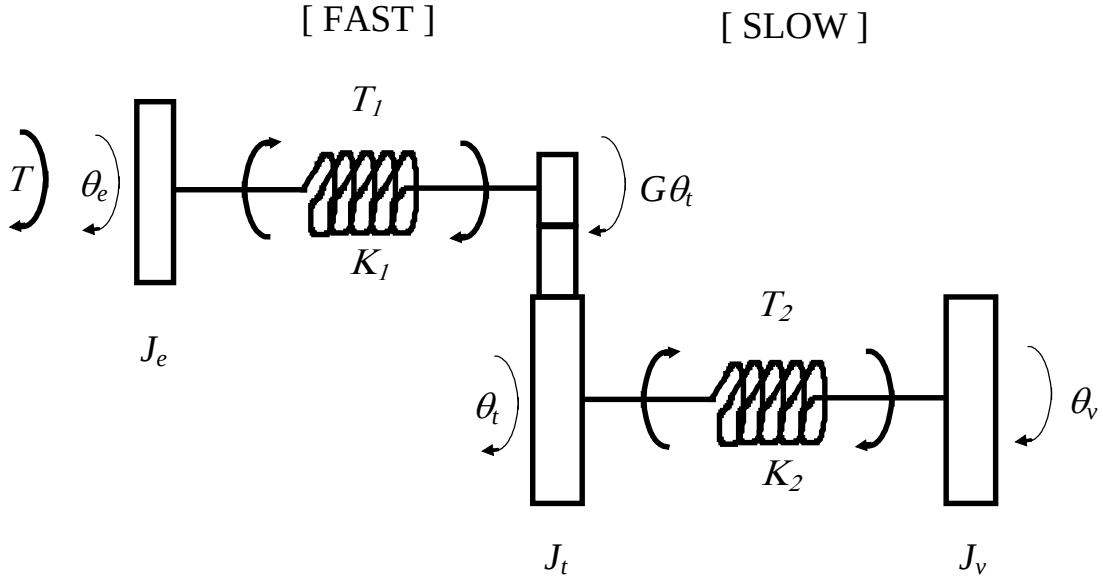
$$G = R / r$$
$$T_a = Fr, \quad T_b = FR$$

so $T_b = GT_a$

Also, as velocity is constant at contact,

$$\dot{\theta}_a = G\dot{\theta}_b \quad \text{and} , \quad \begin{matrix} \ddot{\theta}_a = G\ddot{\theta}_b \\ \theta_a = G\theta_b \end{matrix}$$

In a simple drivetrain model [engine / transmission / vehicle] :



T = input torque, T_1, T_2 = internal torques

$T = J\ddot{\theta}$ applied to :

Hookes law :

Engine : $T - T_1 = J_e\ddot{\theta}_e$ $\quad T_1 = K_1(\theta_e - G\theta_t)$

Transmission : $GT_1 - T_2 = J_t\ddot{\theta}_t$ $\quad T_2 = K_2(\theta_t - \theta_v)$

Wheel : $T_2 = J_v\ddot{\theta}_v$

Substituting for internal torques, T_1 and T_2 :

$$\begin{aligned} T - K_1(\theta_e - G\theta_t) &= J_e\ddot{\theta}_e \\ GK_1(\theta_e - G\theta_t) - K_2(\theta_t - \theta_v) &= J_t\ddot{\theta}_t \\ K_2(\theta_t - \theta_v) &= J_v\ddot{\theta}_v \end{aligned} \qquad (1)$$

This is a *correct* vibration model, but :

(i) It's messy - using G factors in several places
(ii) Different speeds (& vels/accs) make *vibration analysis* less clear

To address (ii), define variables so they have matched steady-state values. Eg, referencing all variables to the *vehicle :*

$$\theta_e^* \equiv \theta_e / G \qquad (\text{ so in rigid motion, } \dot{\theta}_e^* = \dot{\theta}_t = \dot{\theta}_v \ )$$

First two equations (1) are then :

$$\begin{aligned} T - K_1(G\theta_e^* - G\theta_t) &= J_e G\ddot{\theta}_e^* \\ GK_1(G\theta_e^* - G\theta_t) - K_2(\theta_t - \theta_v) &= J_t\ddot{\theta}_t \end{aligned}$$

Even more messy ! - but not if we also define

$$\begin{aligned} J_e^* &\equiv G^2 J_e \\ K_1^* &\equiv G^2 K_1 \end{aligned}$$

The first equation, multiplied by G, is now

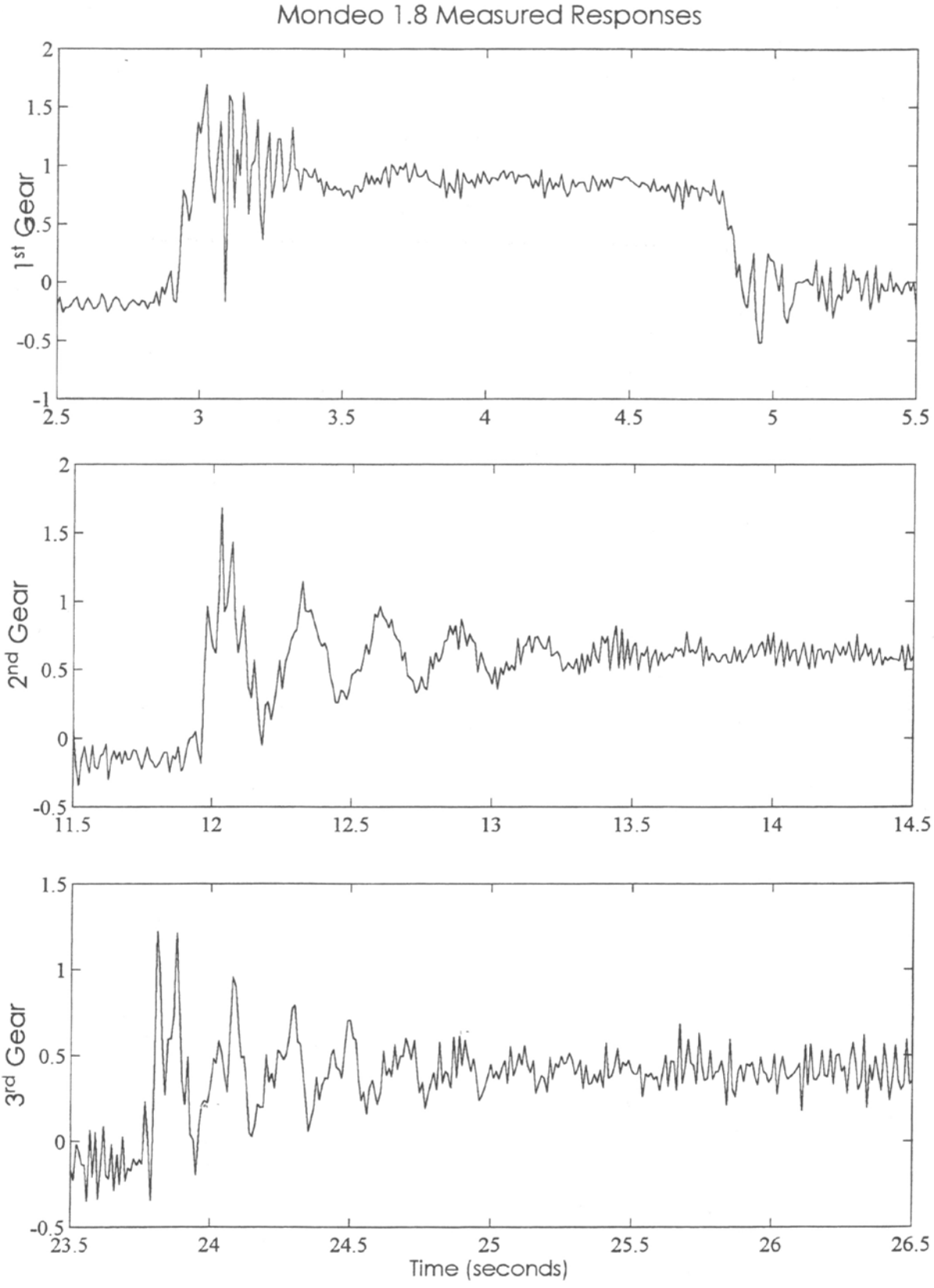
$$GT - K_1^*(\theta_e^* - \theta_t) = J_e^*\ddot{\theta}_e^*$$

and $$K_1^*(\theta_e^* - \theta_t) - K_2(\theta_t - \theta_v) = J_t\ddot{\theta}_t$$

with $$K_2(\theta_t - \theta_v) = J_v\ddot{\theta}_v$$ as before

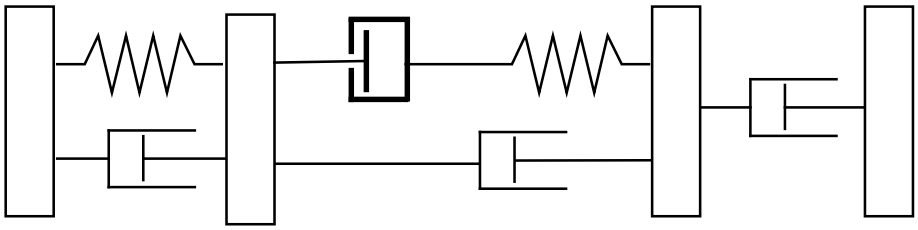
These are general mass / spring equations, with all parameters 'upstream' of the gearbox defined in terms of the gear ratio (damping constants would also be multiplied by G^2). Clearly, **the vibration dynamics will change significantly with gear**.

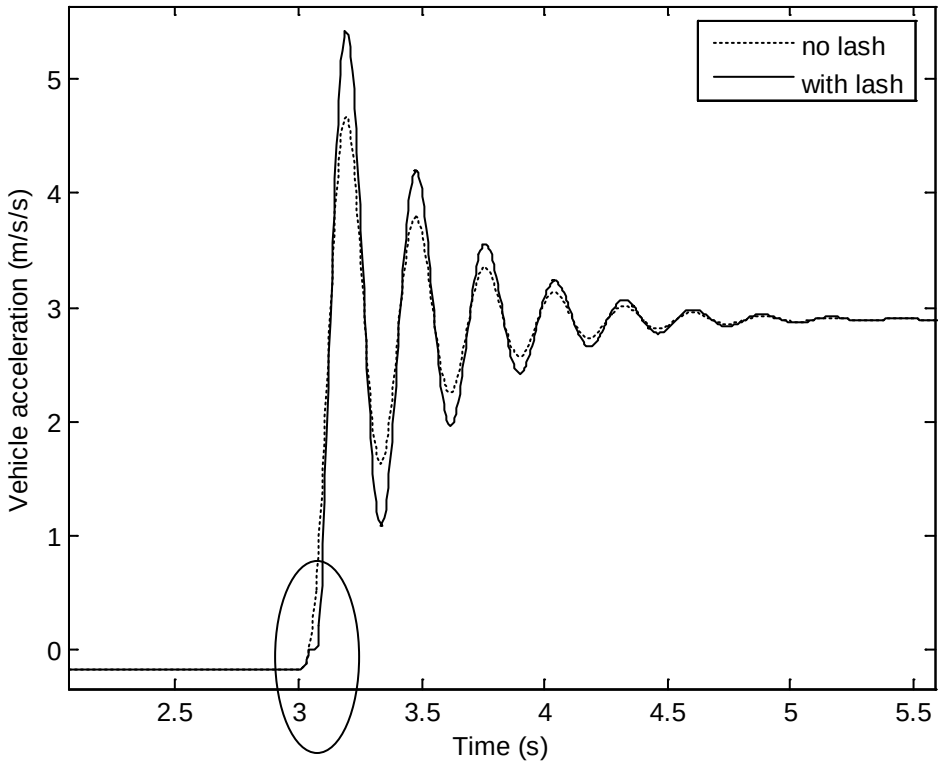
You can see the effect, in terms of altered frequency, and damping rate in these responses, measured from one of the department's old test vehicles. Although there's a fair amount of noise (which you should expect from accelerometers), note how shuffle appears, as a clear second order response.

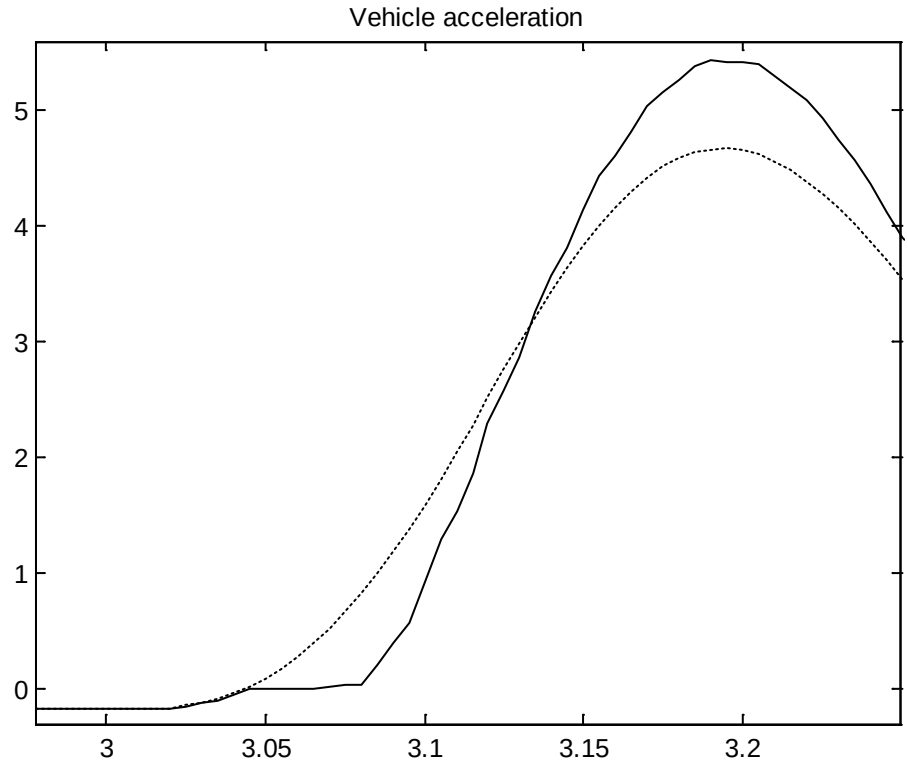


The effect of including lash nonlinearities (also 'clutch lash')

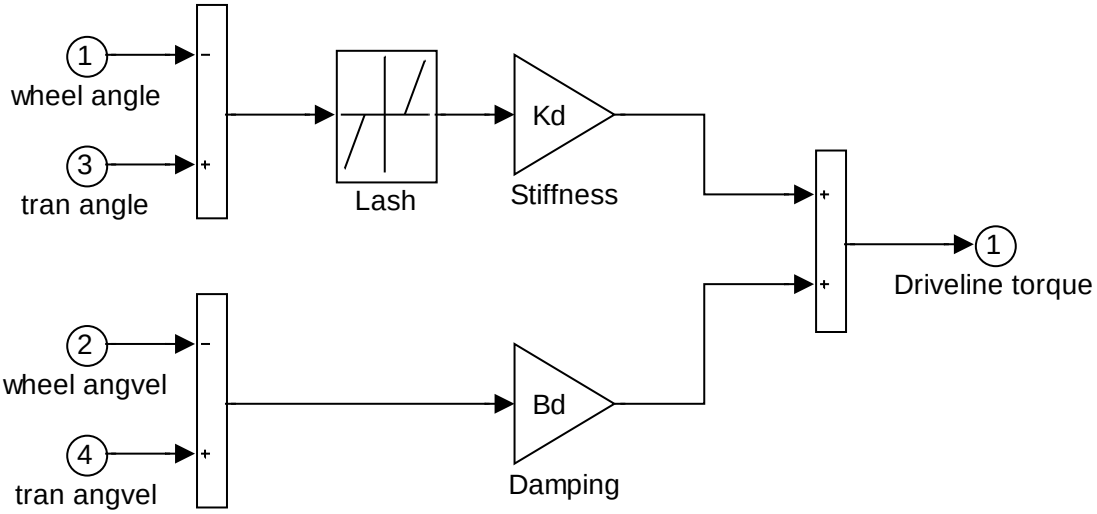
Lash, or free play in the driveline, allows 'upstream' components to accelerate without resistance for a fraction of a second, before reconnecting, and passing on the acquired energy in the form of an impulse. Lash can occur in several places within the driveline, but is modelled only within the axle component in our model, for simplicity.







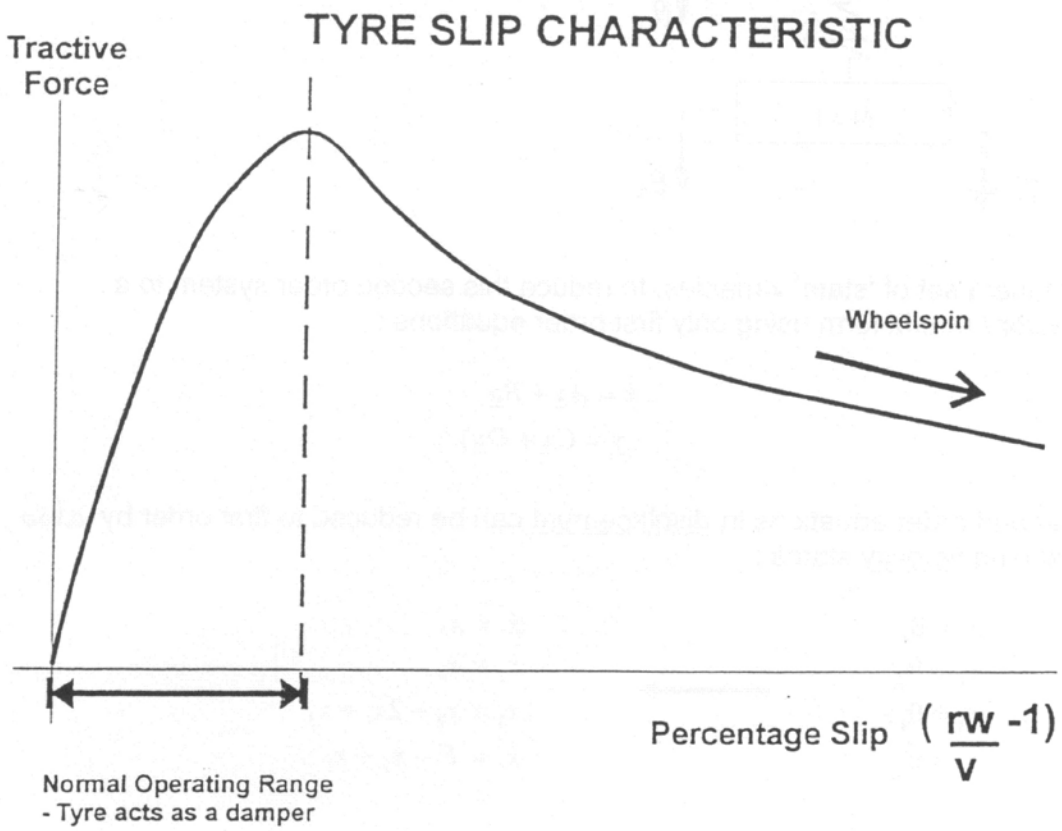
Lash is included in the model using a dead-zone block :



In what ways is this a simplification of reality ? Why not modify the model to make it more accurate ?

An appropriate model for the tyre

Longitudinal tyre force (in acceleration) is generated as a function of rubber compressing as it enters the tyre contact patch, and extending as it leaves. The compression is modelled by a relative speed difference between the contact patch and the road, given by :



5 Free and forced vibration analysis

Linear vs Nonlinear analysis (pros and cons) :

LINEAR SYSTEMS

Pros : Lots of analysis options :

- Eigenvectors / values
- Frequency response
- Simulation studies
- Easy to apply control theory

Cons :

- Less realistic (?)

NONLINEAR SYSTEMS

Pros :

- More realistic

Cons :

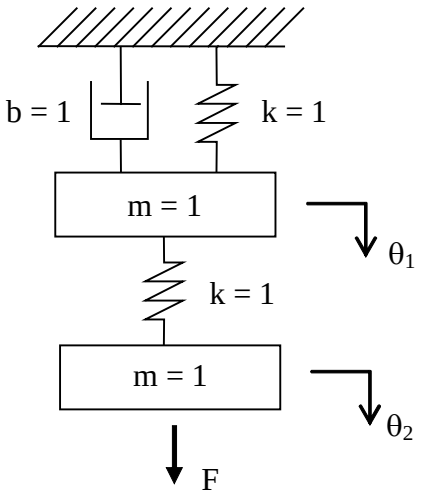
- Can only do simulation studies

What arguments are there for the linear model of the driveline system actually being very realistic ? (Hint : look back at the test vehicle time histories (in different gears), and the 'closer look at driveline model components' above.)

Eigenvector / value (mode) analysis of Linear Systems

The following is largely a repeat of some material we saw earlier, in Section 4. It is included here as a reminder, and we quickly get on, from general cases to analysing the driveline case.

Derivation of a state-space form



Start with second-order differential equations :

$$F - 1(\theta_2 - \theta_1) = 1\ddot{\theta}_2$$

$$1(\theta_2 - \theta_1) - 1\theta_1 - 1\dot{\theta}_1 = 1\ddot{\theta}_1$$

Define a set of 'state' variables, to reduce this second order system to a vector / matrix form using only first order equations :

$$\dot{\underline{x}} = A\underline{x} + B\underline{u}$$
$$(\underline{y} = C\underline{x} + D\underline{u})$$

Second order equations in <u>displacement</u> can be reduced to first order by defining <u>velocity</u> states :

$$\begin{aligned} x_1 &= \theta_1 \\ x_2 &= \theta_2 \\ x_3 &= \dot{\theta}_1 \\ x_4 &= \dot{\theta}_2 \end{aligned} \longrightarrow \begin{aligned} \dot{x}_1 &= x_3 \\ \dot{x}_2 &= x_4 \\ \dot{x}_3 &= x_2 - 2x_1 - x_3 \\ \dot{x}_4 &= F - x_2 + x_1 \end{aligned}$$

Filling in the matrices, *A* and *B* :

displacements
velocities

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 1 & -1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} F \qquad \text{-------- (Eqn 1)}$$

$$\dot{\underline{x}} = A\ \underline{x} + B\ \underline{u}$$

Matrix *A* has very useful properties . . . .

Modal Vibrations

The free vibration of the masses (F=0) can be described by a summation of modal components having a predictable form :

$$\underline{\theta}(t) = \mathrm{Re}\{\underline{u}_1 e^{\lambda_1 t} + \underline{u}_2 e^{\lambda_2 t} + \ldots + \underline{u}_n e^{\lambda_n t}\}$$

Where $\underline{\theta}(t)$ is the 2x1 displacement vector, $\underline{u}_k$ is a complex constant 2x1 vector, and λ_k is a complex scalar.

Within each 'modal component' :

$$\underline{\theta}(t) = \underline{u}_1 e^{\lambda_1 t}, \qquad \text{so} \qquad \dot{\underline{\theta}}(t) = \lambda_1 \underline{u}_1 e^{\lambda_1 t}$$

therefore $\underline{x}(t) = \underline{v}_1 e^{\lambda_1 t}$, where $\underline{v}_1 = \begin{bmatrix} \underline{u}_1 \\ \lambda_1 \underline{u}_1 \end{bmatrix}$

Now, if F=0, from Eqn 1 :

$$\lambda_1 \underline{v}_1 e^{\lambda_1 t} = A\underline{v}_1 e^{\lambda_1 t}$$

Therefore $$\lambda_1 \underline{v}_1 = A\underline{v}_1$$

so λ_1 is an <u>eigenvalue</u> of A, and $\underline{v}_1$ is the corresponding <u>eigenvector</u>.

For the spring / mass example above, the four eigenvalues are

λ_1 = -0.35 + 1.5j]
λ_2 = -0.35 - 1.5j]
λ_3 = -0.15 + 0.63j]
λ_4 = -0.15 - 0.63j]

Note that the eigenvalues come in <u>pairs</u>, with each pair describing a <u>mode</u> of vibration.

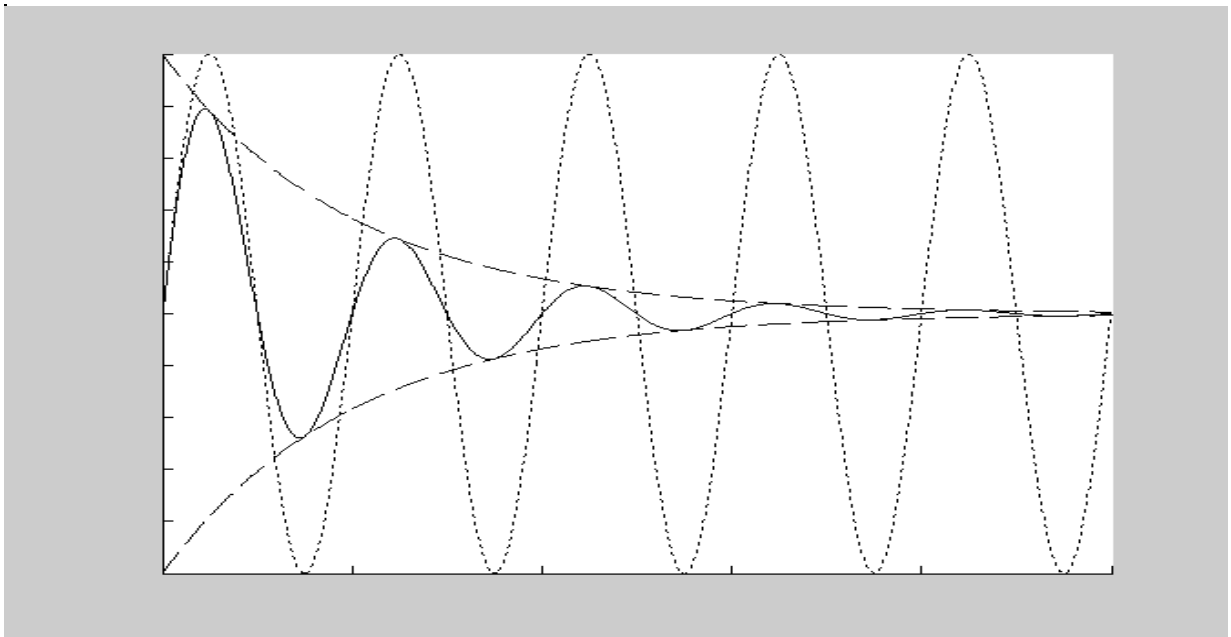
The eigenvalues equate to the system poles, and pole pairs can be written generally as :

$$\lambda_{1\&2} = \sigma \pm j\omega$$

where σ is the modal damping factor, and ω is the damped natural frequency of the mode. To see what each single term in $\underline{\theta}(t) = \mathrm{Re}\{\underline{u}_1 e^{\lambda_1 t} + \underline{u}_2 e^{\lambda_2 t} + \ldots + \underline{u}_n e^{\lambda_n t}\}$ 'looks like', re-substitute λ and use Euler's formula for the complex exponent :

$$\underline{u}_1 e^{\lambda_1 t} = \underline{u}_1 e^{(\sigma+\omega i)t} = \underline{u}_1 e^{\sigma t} e^{i\omega t} = \underline{u}_1 e^{\sigma t}\left(\cos(\omega t) + i\sin(\omega t)\right)$$

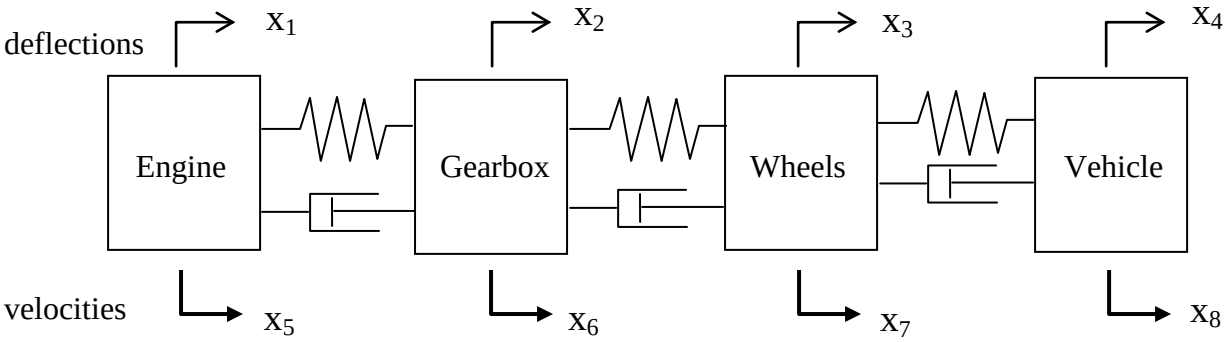
σ should be negative if your model is correct, and it relates to a physical system, because this determines the 'envelope' of the response, as a decaying exponential (shown below in dashed lines). ω then gives the frequency of the sinusoid component (dotted lines), (cos ωt + isin ωt) and the solid line shows the combination of the two.



This is then multiplied by the complex constants, $\underline{u}_i$,which affect the magnitude of the response (in this case the magnitude of θ_1 compared with θ_2) and the relative phase (eg 90° phase difference would have the θ_1 at 0 at the same time as θ_2 is at its peak).

Eigenstructure of the four mass Driveline model :

The order of the states is important when it comes to interpreting eigenvectors. Here we choose the (rotational) deflections first, followed by the velocities :



NB : To achieve this ordering of states for your Simulink model, assign outports (at the top level of the model) to each of the required states, in the order shown above. Because the outputs correlate with the states, when you carry out further analysis (eg via **`linmod`**) on the model, Simulink will choose the states in the same order as the outputs. (Without outputs, the states are chosen in an arbitrary order, depending on when the integrators were 'dropped' into the Simulink model.) You can check that you've got an exact match of states = outputs by checking the C matrix you get from linmod (which should be the identity matrix).

When the model is set up correctly, it is very easy to extract the Eigenstructure in Matlab :

```
[A,B,C,D] = linmod('drive4');
[V,D] = eig(A);
```

Eigenvalues (shaded), and their associated Eigenvectors (tabled).

Note how each row of the Eigenvector matrix relates to a state, and each column relates to an Eigenvalue.

	λ_1 = 0	λ_2 = -779.5	λ_3 = -86.9+366.4i	λ_4 = -86.9-366.4i
x1	0	0.0000	-0.0000 - 0.0001i	-0.0000 + 0.0001i
x2	0	-0.0000	0.0006 + 0.0026i	0.0006 - 0.0026i
x3	0	-0.0013	0.0000 - 0.0000i	0.0000 + 0.0000i
x4	1.000	0.0000	-0.0000 - 0.0000i	-0.0000 + 0.0000i
x5	0	-0.0000	0.0269 + 0.0007i	0.0269 - 0.0007i
x6	0	0.0129	-0.9996	-0.9996
x7	0	0.9998	0.0059 + 0.0116i	0.0059 - 0.0116i
x8	0	-0.0150	0.0003 - 0.0003i	0.0003 + 0.0003i

	λ_5 = -2.40 + 22.26i	λ_6 = -2.40- 22.26i	λ_7 = -1.4676e-006	λ_8 = 1.4676e-006
x1	-0.0035 - 0.0329i	-0.0035 - 0.0329i	0.5000	-0.5000
x2	-0.0039 - 0.0284i	-0.0039 - 0.0284i	0.5000	-0.5000
x3	-0.0068 + 0.0040i	-0.0068 + 0.0040i	0.5000	-0.5000
x4	0.0005 + 0.0038i	0.0005 + 0.0038i	0.5000	-0.5000
x5	0.7403	0.7403	-0.0000	-0.0000
x6	0.6412 - 0.0186i	0.6412 - 0.0186i	-0.0000	-0.0000
x7	-0.0720 - 0.1618i	-0.0720 - 0.1618i	-0.0000	-0.0000
x8	-0.0848 + 0.0025i	-0.0848 + 0.0025i	-0.0000	-0.0000

Which Eigenvalues relate to shuffle and which to rattle ? What modes do the other Eigenvalues represent ? Calculate damped natural frequencies, damping ratios and settling times for shuffle and rattle.

Dividing by the largest element within an appropriate set of four eigenvectors, we can derive the relative motion between masses, in the rattle mode :

```
 0.0269 + 0.0007i      / -0.9996      =     -0.0269 - 0.0007i
-0.9996                / -0.9996      =      1.0000
 0.0059 + 0.0116i      / -0.9996      =     -0.0059 - 0.0116i
 0.0003 - 0.0003i      / -0.9996      =     -0.0003 + 0.0003i
```

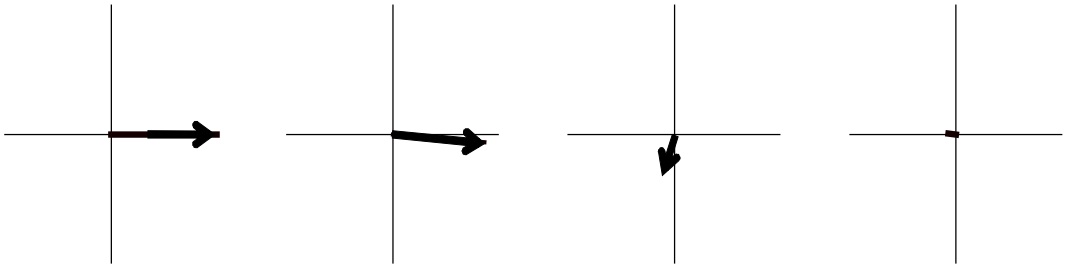
and in the shuffle mode :

```
  0.7403               / 0.7403       =      1.0000
  0.6412 - 0.0186i     / 0.7403       =      0.8661 - 0.0251i
 -0.0720 - 0.1618i     / 0.7403       =     -0.0972 - 0.2185i
 -0.0848 + 0.0025i     / 0.7403       =     -0.1146 + 0.0034i
```

It is easier to visualise this if you plot the eigenvector components as real vs imag ‘phasors’ arranged like the masses in the model :

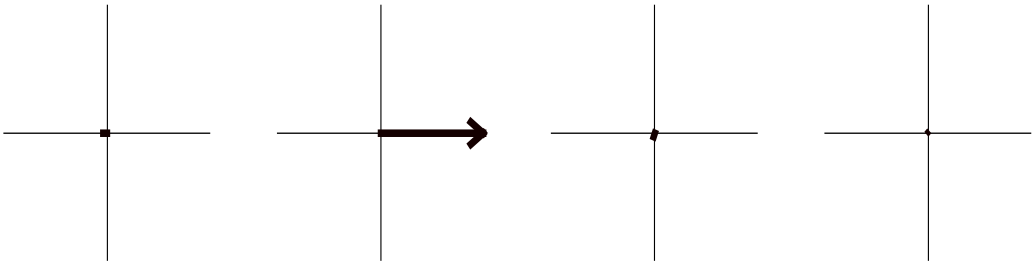
Mode 1 ‘Shuffle’

```
1.0000      0.8661-0.0251i    -0.0972-0.2185i    -0.1146+0.0034i
```



Mode 2 ‘Rattle’

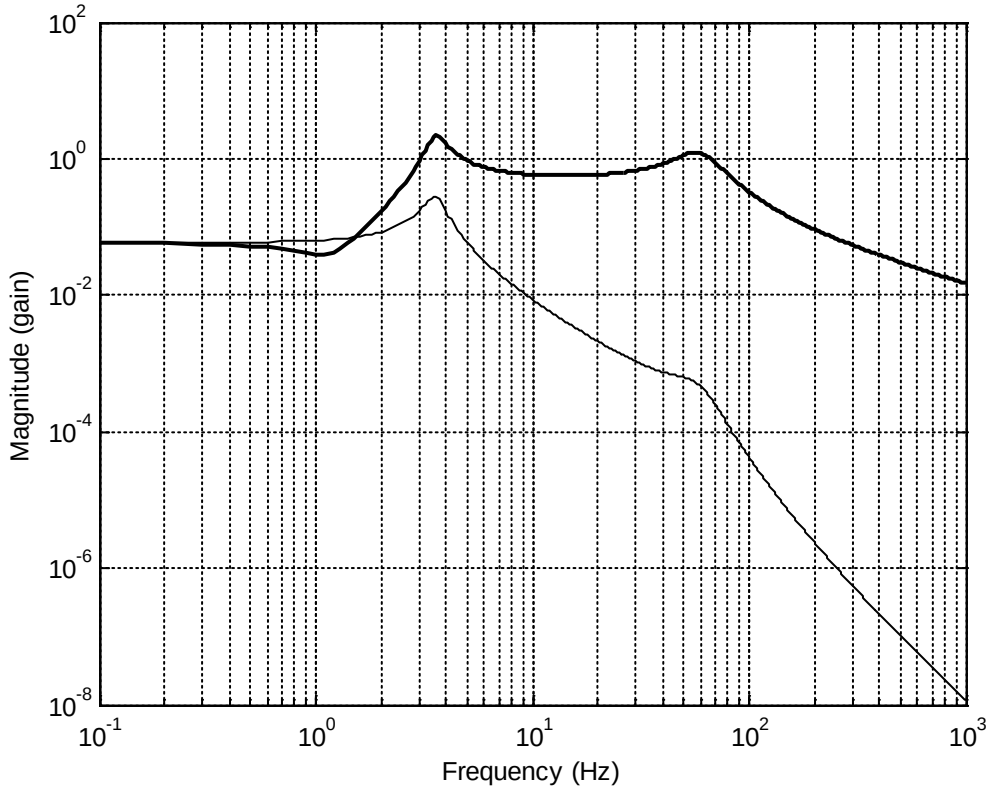
```
-0.0269-0.0007i       1.0000       -0.0059-0.0116i   -0.0003+0.0003i
```



Frequency Response

For the Eigenstructure analysis we assigned outports to each of the states. Now look at the frequency response (Bode plot) between the engine torque input, and the vehicle acceleration, and transmission (gearbox) acceleration outputs. (Assign inports and outports at the top level of the model, appropriately.). Then

```
[A,B,C,D] = linmod('drive4');
sys = ss(A,B,C,D);
f = [0.1:0.1:1000]';
[mag,phase] = bode(sys,f*2*pi);
mag = squeeze(mag)';
loglog(f,mag);
grid on;
```



Which is the transmission acceleration response, and which is that of the vehicle acceleration ?