Suspension Systems and Control

Matlab Case Study for Signals and Systems

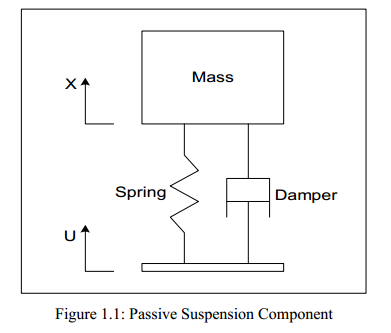
Suspension systems are an invisible phenomenon of our daily lives. Although they are built into nearly every vehicle we use, we only notice their existence when they are broken - as anyone who has driven on a poorly maintained road in an old car can attest. These essential systems incorporate elements of physics, material engineering, signal processing, and control systems in order to ensure that your daily transit is a comfortable one.

In this case study, you will simulate a simple suspension system and analyze its behavior. You will study the relationship between physical properties (such as spring constant and damping coefficient), physical behavior (such as how well the system absorbs shocks), the transfer function of the system, and the location of zeros and poles. You will use these observations to simulate your own suspension system and choose its properties to absorb shocks from potholes and reduce noise from bumpy roads.

You will also explore the use of active suspension systems, which use sensors and hydraulic actuators rather than springs and damping. You will simulate an active suspension system and learn how to implement state feedback to alter the characteristics of a closed-loop transfer function.

# Passive Suspension Systems

A passive vehicle suspension system is a system designed to absorb shocks and vibrations using mechanical devices such as springs which require no sensors or controls. In its simplest form, it can be modeled as a mass *m* held up by an ideal spring with spring constant *k*, and a damper with damping coefficient *b*. As the vehicle travels along the road, a displacement *u(t)* is applied to one end of the spring, compressing it. This in turn moves the vehicle body with displacement *x(t)*. This system is shown below:



(Although some versions of this system include the vehicle wheel as an additional spring, we imagine that the wheel is perfectly rigid and rolls smoothly without slipping or deformation) Writing out Newton’s second law for this system, we arrive at:

Which we can reformulate as an input-output equation:

# Passive Case Study

Using the *passive\_suspension.m* file as a template, complete the following tasks:

* Use MATLAB to simulate the input-output equation shown above using any method of your choice. (Manual iterations, ode45, converting to a difference equation, state-space modeling, or even Simulink). Your model should use zero initial conditions and use the roadSurface vector as its input. It should output x(t): the position of the vehicle body. The output vector length should match roadSurface.
* Use the *animateCar()* function to visualize the results of your simulation. Does the passive suspension system effectively absorb shocks and filter out the bumpiness of the road? Record any observations in your writeup.
* Follow the instructions in section 2 of the *passive\_suspension.*m script to calculate the transfer function of the system and use it to plot the poles and zeros. What do you notice about the location of poles and zeros? Record any observations in your writeup.
* Experiment with your model; change the spring constant and damping coefficient and note the effects of these changes on the animation and on the pole/zero map. Record any observations in your writeup.
* Follow the instructions in section 3 of the *passive­\_suspension.m* script to devise a suspension system that will work effectively for a vehicle that varies in mass. Record your results in your writeup.

# Active Suspension Systems

You may have noticed a problem with the passive suspension system; because the spring constant and damping coefficient of the system can’t change, but the mass of the vehicle can, it is impossible to ensure the system will always be critically damped. A spring might be too strong for an unloaded vehicle but too weak for a loaded one. Active suspension systems use sensors and hydraulic actuators to reduce shocks to the vehicle, rather than leaving it all up to springs. Though they are more complicated systems, they are also more robust and adaptable.

Consider a modified version of our previous model with a piston attached. This piston runs between the wheel and the vehicle body, alongside the spring and damper. The piston can provide a variable force *f* to the vehicle body based on input from the vehicle’s sensors. For simplicity, we’ll assume the vehicle’s sensors can perfectly measure its height above the ground, its vertical acceleration, and its vertical velocity.

A picture containing clock

Description automatically generated

We slightly modify our force equation:

Resulting in the input-output equation:

# Active Case Study

Using the *active\_suspension.m* file as a template, complete the following tasks:

* Calculate the open-loop transfer function of the system – the transfer function that results from applying no feedback to the controller at all – and analyze its pole map. This should be very similar to your work in the previous section.
* Complete the piston\_controller function to devise a state-feedback controller. Test your controller with a variety of vehicle masses as you did in the previous section.
* Calculate the transfer function of your controller and the closed-loop transfer function of the entire active suspension system. Map the poles and zeros of the closed loop system. Record any observations in your writeup.

# What to turn in

* Your completed passive\_suspension.m file.
* Your completed active\_supension.m file.
* Your completed piston\_controller.m file.
* Any additional functions or scripts you wrote
* A brief writeup summarizing your work and observations on each section.