Modelling Exercise

August 12, 2013

Contents

1 Introduction

The aim of this exercise is to develop and implement a model of a physical system that can be controlled using a PID within the MARTe real-time framework, and to test the effectiveness of such a controller using a variety of reference waveforms.

2 The Model

2.1 The Spring-Mass Model

The system consists of a body of mass m_1 connected to a fixed wall by a spring with spring constant k_1 and a damper with damping constant c_1. There is a piston that exerts a force $F(t)$ on the mass, which can be treated as a system input. There is a second mass attached to the first by another spring and damper with constants k_2 and c_2 respectively.

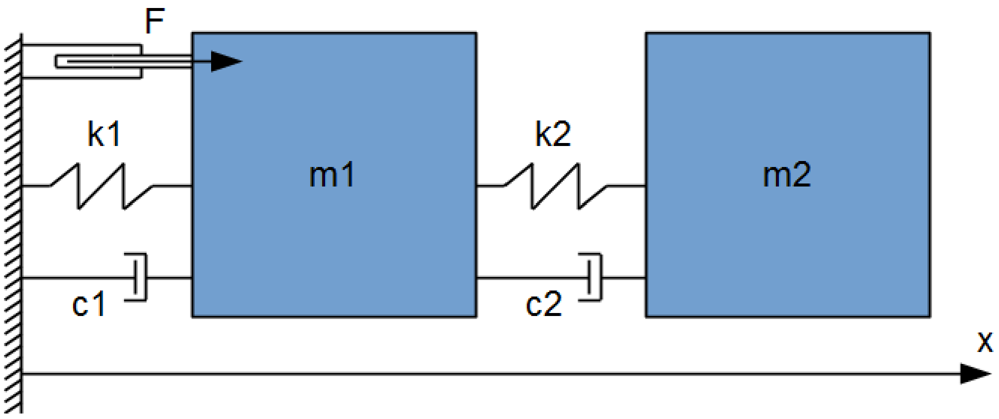

Figure 1: The mechanical model

2.2 Mathematical Model

Considering the forces exerted on the masses by the springs and dampeners, and also the force exerted by the piston, Newton's second law can be used to form two differential equations that describe the acceleration of the masses:

$$m_1\ddot{x_1} = -k_1 s_1(t) + k_2 s_2(t) - c_1\dot{s_1}(t) + c_2\dot{s_2}(t) + F(t) \tag{1}$$

$$m_2\ddot{x_2}(t) = -k_2 s_2(t) - c_2\dot{s_2}(t) \tag{2}$$

where x_n are the positions of the masses relative to the wall and s_n are the extensions of the springs. The spring extensions and their derivatives can be expressed as

$$x_1(t) = s_1(t) + L_1 \qquad x_2(t) = L_1 + L_2 + W_1 + s_1(t) + s_2(t) \tag{3}$$
$$\dot{x_1}(t) = \dot{s_1}(t) \qquad \dot{x_2}(t) = \dot{s_1}(t) + \dot{s_2}(t) \tag{4}$$
$$\ddot{x_1}(t) = \ddot{s_1}(t) \qquad \ddot{x_2}(t) = \ddot{s_1}(t) + \ddot{s_2}(t) \tag{5}$$

where L_1 and L_2 are the rest lengths of the springs and W_1 is the width of the first mass. Using equations (3-5) to substitute x_n for s_n in equations (1,2) yields

$$m_1\ddot{s_1}(t) = -k_1 s_1(t) + k_2 s_2(t) - c_1\dot{s_1}(t) + c_2\dot{s_2}(t) + F(t) \tag{6}$$

$$m_2\ddot{s_2}(t) = \frac{m_2}{m_1}k_1 s_1(t) + \frac{m_2}{m_1}c_1\dot{s_1}(t) - \left(\frac{m_1+m_2}{m_1}\right)k_2 s_2(t) - \left(\frac{m_1+m_2}{m_1}\right)c_2\dot{s_2}(t) - \frac{m_2}{m_1}F(t) \tag{7}$$

which fully describe the extension of each spring at a time (t). It would be possible to express the system using either the mass positions x_n or the spring extensions s_n as state variables, but here the extensions were used. Equations (6) and (7) can be expressed more elegantly in a matrix representation:

$$\frac{d}{dt}\begin{bmatrix} s_1 \\ \dot{s_1} \\ s_2 \\ \dot{s_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_1}{m_1} & -\frac{c_1}{m_1} & \frac{k_2}{m_1} & \frac{c_2}{m_1} \\ 0 & 0 & 0 & 1 \\ \frac{k_1}{m_1} & \frac{c_1}{m_1} & -\left(\frac{m_1+m_2}{m_1 m_2}\right)k_2 & -\left(\frac{m_1+m_2}{m_1 m_2}\right)c_2 \end{bmatrix}\begin{bmatrix} s_1 \\ \dot{s_1} \\ s_2 \\ \dot{s_2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m_1} \\ 0 \\ -\frac{1}{m_1} \end{bmatrix} F(t) \tag{8}$$

3 MARTe Implementation

3.1 Solution using the Euler method

The SpringMassModel GAM is implemented using the (forwards) Euler method:

$$s_n(k+1) = s_n(k) + \dot{s_n}(k)\triangle t \tag{9}$$

$$\dot{s_n}(k+1) = \dot{s_n}(k) + \ddot{s_n}(k)\triangle t \tag{10}$$

where k denotes discretized time and the $\ddot{s_n}$ can be taken from equation (8).

3.2 Collisions

To make the model behave more realistically, the model GAM checks for collisions between the mass and the wall and also between the two masses. It does this by comparing the spring extensions to the spring rest lengths. If the Euler method calculates that either spring is compressed to beyond the spring rest length, then there must be a collision. The code checks for a collision between the first mass and the wall by checking for$-s_1 < L_1$, and similarly for a collision between masses by checking for $-s_2 < L_2$. If a collision has occurred, then the mass velocities are changed accordingly as detailed in section A.1.

3.3 Configuration File

The configuration file for the spring-mass system includes the following GAMs:

TimeInputGAM the time source

WaveformGenerator the reference wave for the system to follow

PIDGAM the PID controller

SpringMassModel the model of the system

WebStatisticGAM provides real-time information about the signals used by the MARTe framework

DataCollectionGAM stores measurements for offline analysis

PlottingGAM provides a real time plot of specified signals

The SpringMassModel GAM takes the current time and requested force (as decided from the PID GAM) as inputs and returns the spring extensions, mass positions, spring extension rates and piston force. It also saturates the force, using the maximum and minimum limits specified in the configuration file.

The parameters and initial conditions of the spring-mass model are given in the configuration file. If a parameter has not been specified, it is assigned a default value and a message is sent to the logger. The parameters used by the GAM are listed in section A.2.

3.4 Spring-Mass Model Animation

The SVG vector image file made for the spring-mass model is a modified version of the water tank SVG. It begins by getting the model parameters (namely spring rest length, mass width and spring stiffness constants), and then uses these to draw an image of the model. It detects which spring is of a higher stiffness and emboldens it. It then enters a loop that gets the spring extensions and piston force and adjusts the image accordingly.

When dealing with particularly long/short springs or masses, it may be necessary to adjust the `METRE_TO_PIXEL` variable in order for the image to be a useful representation of the system.

4 Results

4.1 Free Model

The time evolution of a model with initial extensions but no reference or control was used to examine collisions. The parameters as given in section A.3 were chosen to ensure collisions between both the masses and also with the wall. As can be seen in figure 2, the case with inelastic collisions experiences sudden decreases in total energy, whereas the case with elastic collisons has no such decrease. In both cases, the energy tends to zero due the presence of dampers. These two observations indicate that the model is working correctly.

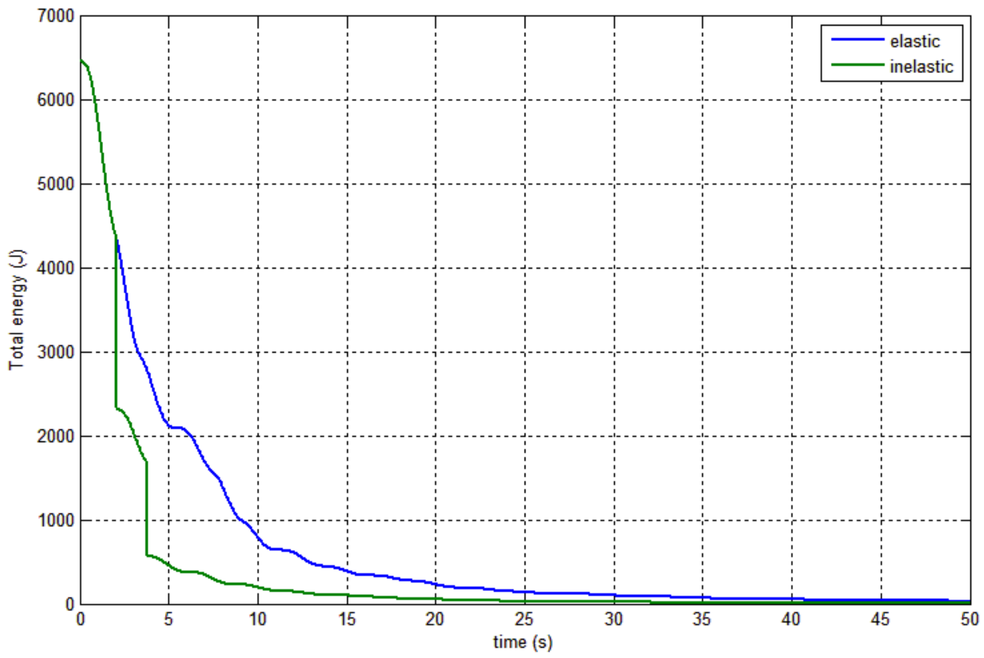

Figure 2: The time evolution of the energy of the free system for perfectly elastic ($C_R = 1$) and perfectly inelastic ($C_R = 0$) collisions. Energy was calculated as $E_{total} = E_{kinetic} + E_{potential} = \frac{1}{2}m_1\dot{x_1}^2 + \frac{1}{2}m_2\dot{x_2}^2 + \frac{1}{2}k_1s_1^2 + \frac{1}{2}k_2s_2^2$

4.2 PID Control

In the following examples, the position of the second mass was controlled using a PID controller. This was tuned by using the Ziegler-Nichols to get an estimate of a value, and then tuned manually in order to improve the response. This was generally not simply a light touch, so it should be noted that the Ziegler-Nichols method does not seem suited to this particular plant. Two sets of PID tunings were used: a tuning based on the classic Ziegler-Nichols method with $K_p = 6.1$, $K_i = 4.1$, and $K_d = 6.3$, and a second based on the "no overshoot" Ziegler-Nichols method with $K_p = 2.0$, $K_i = 4.1$, and $K_d = 17.0$. There is no doubt room for improvement on these, and it ought to be noted that these tunings go against the design philosophy that most of the gain should be due to the proportional term and the derivative term should be kept small, but in this case these seem to give good responses. The parameters of the model used for this section are given in section A.3.

4.2.1 Step Reference

The reference used for this was fixed at 6m. From figure 3 it can be seen that the higher value of K_p used in the classic tuning results in a decreased rise time, but the higher derivative par used in the no overshoot tuning has a shorter settling time. Of the two tunings, the latter is probably more suited to this particular reference.

4.2.2 High Frequency Sinusoidal Reference

The reference wave used for this section was a sinusoidal wave centered on 6m with amplitude 1m and period 10s. Figure 4 suggests that higher proportional gain is useful for forcing the system to follow a sinusoidal reference wave. There is some phase difference present and also a marked difference in the amplitude of the displacement for both tunings. It may be possible to rectify this by adjusting the tunings so that proportional gain is higher so that rise time is decreased, though the nature of the PID controller means that a sinusoidal reference wave cannot be *perfectly* matched.

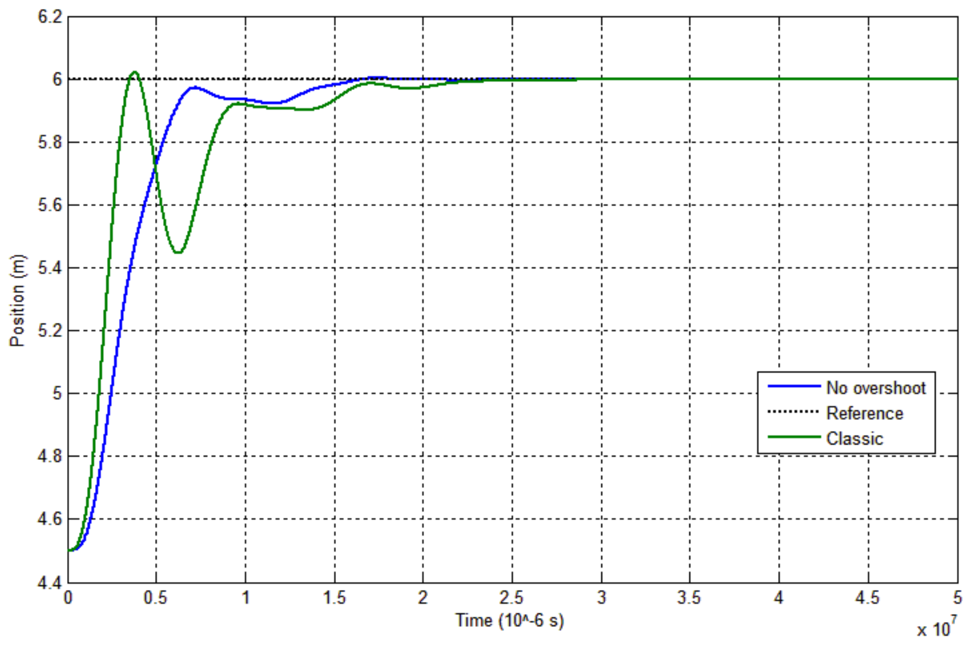

Figure 3: Response in the position of the second mass for a step reference wave.

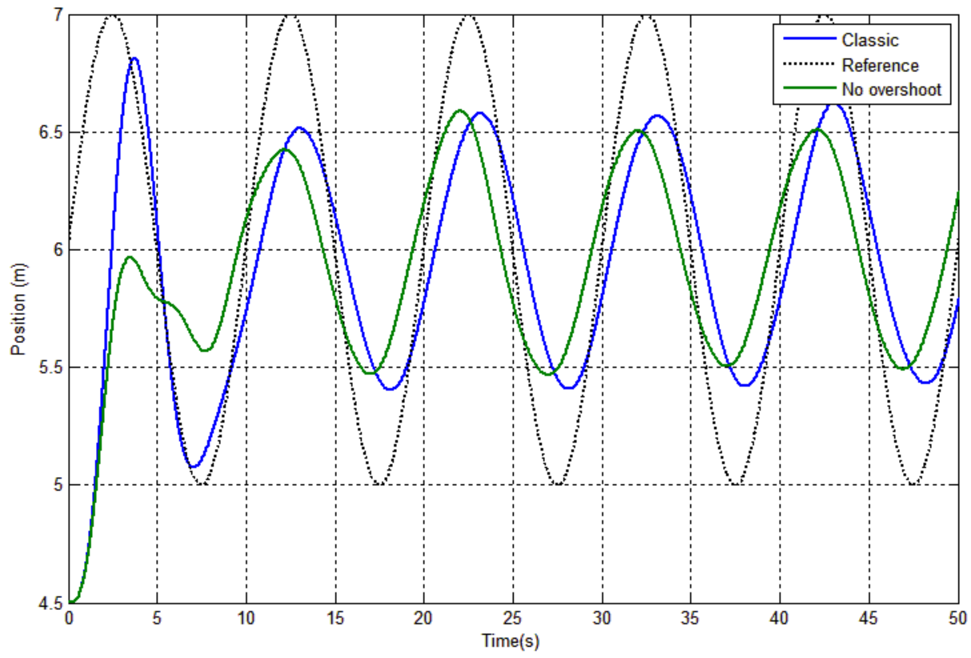

Figure 4: Response in the position of the second mass for a sinusoidal reference wave.

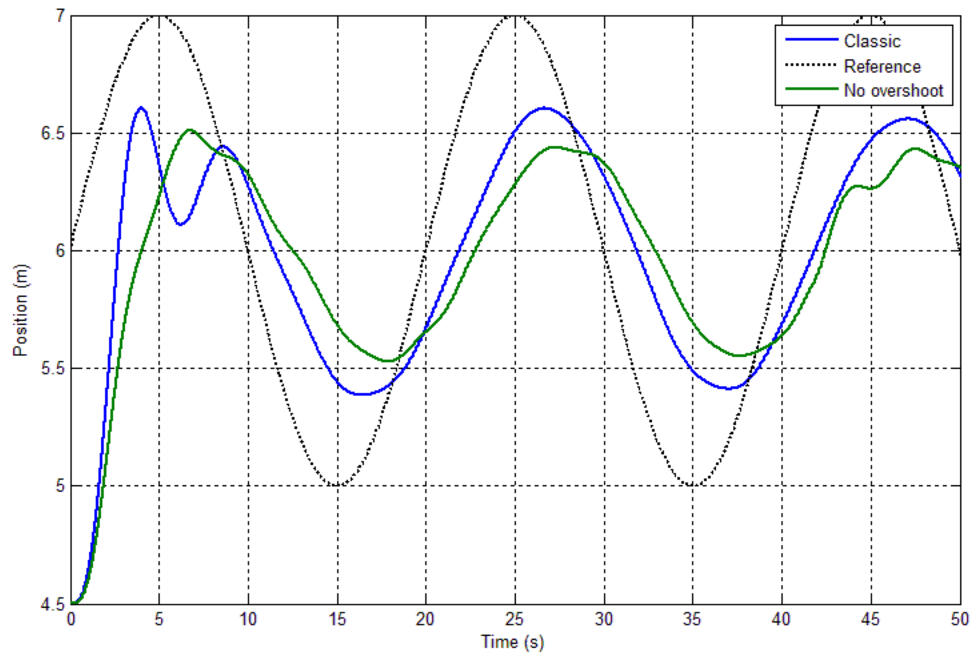

Figure 5: Response in the position of the second mass for a low frequency sinusoidal reference wave.

4.2.3 Low Frequency Sinusoidal Reference

The reference wave used for this section was a sinusoidal wave centered on 6m with amplitude 1m and period 20s. The low frequency sinusoidal wave in figure 5 is similar to the high frequency reference in that each tuning method is still unable to reach the full amplitude of the wave and there is also a phase shift. Of the two methods, the classic is better suited, since it has higher amplitude and fewer curve disturbances.

4.2.4 Triangle Reference

The reference wave used for this section was a triangle wave centered on 5.5m with amplitude 0.5m and period 10s.The classic tuning seems more suited to this kind of reference wave as can be seen in figure 6 due to a decreased rise time. The no overshoot method closely matches the rising and falling edges of the triangle wave but dips inwards when it should peak.

5 Conclusions

Since the plant is a second order system, it is prone to oscillations in its output (the position of the second mass). This is rectified by using a large derivative gain in the PID, though this is generally avoided in a real-world control system. The settling time would be very large without the the derivative term.

Of the two tuning systems used, the classic method is better at following time-varying reference waveforms due to a higher proportional gain, but the no-overshoot method gives a better step response.

Points for Further Study

More plant parameters could be tested and compared. For example, it could be expected that a plant with a very large spring 2 stiffness constant will react similarly to a system with only a single mass.

The PID tuning could be improved, or a model-based approach could be used to correct for the steady-state error (which here is compensated for using the integral part of the PID).

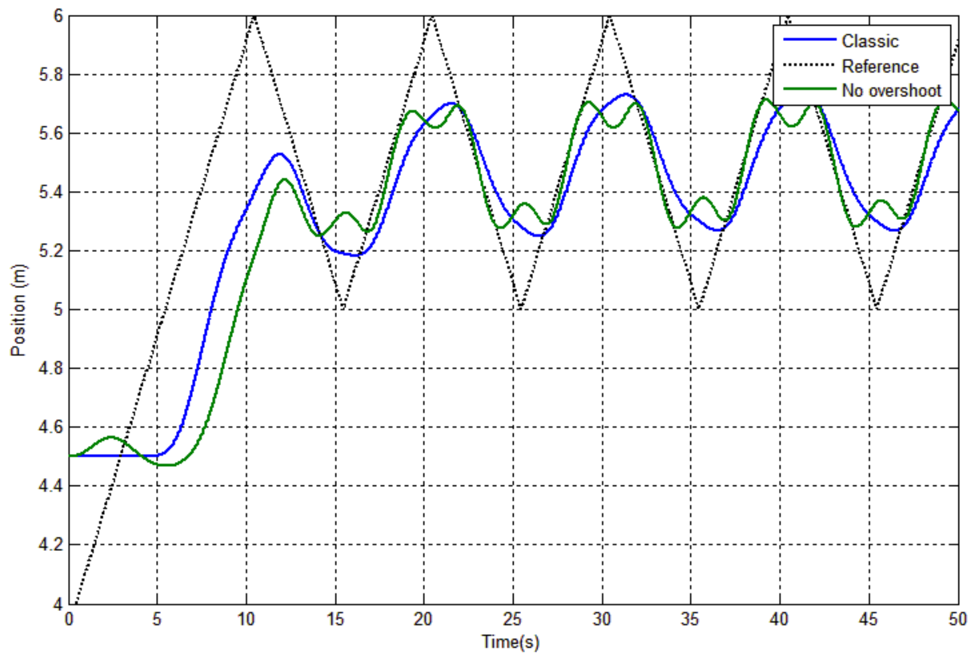

Figure 6: Response in the position of the second mass for a triangle reference wave.

A Appendix

A.1 Elastic and Inelastic Collisions

In a one dimensional collision, the final velocities v_a,v_b of two colliding bodies of masses m_a,m_b traveling at velocities u_a,u_b are

$$v_a = \frac{m_a u_a + m_b u_b + m_b C_R(u_b - u_a)}{m_a + m_b} \tag{11}$$

$$v_b = \frac{m_a u_a + m_b u_b - m_a C_R(u_b - u_a)}{m_a + m_b} \tag{12}$$

where C_R is the coefficient of restitution, defined as

$$C_R = \frac{\text{Relative speed after collision}}{\text{Relative speed before collision}} = \frac{v_b - v_a}{u_a - u_b} \tag{13}$$

When $C_R = 1$ the collision is perfectly elastic (kinetic energy conserved), and when $C_R = 0$ the collision is perfectly inelastic (maximum amount of kinetic energy is lost). Most real world collisions are partially inelastic collisions and have $0 < C_R < 1$.

Mass1-Wall Collision

In this case, it is assumed that the wall is fixed. This is equivalent to making it very massive, such that $m_{wall} \gg m_1$ which leads to $\frac{m_1}{m_{wall}} \approx 0$. Substituting the appropriate quantities into equations (11,12) and using the fact that the wall is stationary before the collision results in:

$$\dot{s_1}(\text{final}) = \frac{m_1\dot{s_1} - C_R m_{wall}\dot{s_1}}{m_1 + m_{wall}} = \frac{\frac{m_1}{m_{wall}}\dot{s_1} - C_R\dot{s_1}}{\frac{m_1}{m_{wall}} + 1} \approx -C_R\dot{s_1}$$

$$v_{wall} = \frac{m_1\dot{s_1} + C_R m_1 \dot{s_1}}{m_1 + m_{wall}} = \frac{\frac{m_1}{m_{wall}}(\dot{s_1} + C_R\dot{s_1})}{\frac{m_1}{m_{wall}} + 1} \approx 0$$

where $\dot{s_1}$ represents the initial velocity of the first mass. In order to conserve momentum and kinetic energy, the velocity of the second mass should not change when this collision takes place, so:

$$\dot{x_2}(\text{final}) = \dot{x_2}(\text{initial})$$
$$\dot{s_1}(\text{final}) + \dot{s_2}(\text{final}) = \dot{s_1}(\text{initial}) + \dot{s_2}(\text{initial})$$
$$\dot{s_2}(\text{final}) = \dot{s_1}(\text{initial}) + \dot{s_2}(\text{initial}) - \dot{s_1}(\text{final})$$

The SpringMassModel GAM applies this correction whenever a mass1-wall collision is detected.

Mass1-Mass2 Collision

Again manipulating equations (11,12), the following equations can be derived:

$$\dot{s_1}(\text{final}) = \frac{m_1\dot{s_1} + m_2(\dot{s_1} + \dot{s_2}) + C_R m_2 \dot{s_2}}{m_1 + m_2}$$

$$\dot{s_2}(\text{final}) = \frac{m_1\dot{s_1} + m_2(\dot{s_1} + \dot{s_2}) - C_R m_1 \dot{s_2}}{m_1 + m_2} - \dot{s_1}(\text{final}) = -C_R\dot{s_2}$$

A.2 SpringMassModel GAM Parameters

The following is a list of the SpringMassModel parameters that can be specified in the configuration file.

`SVGFileLocation` - location of the SVG file to be used if animation is enabled.
`AnimationEnabled` - set equal to 1 to enable animation in the SpringMassModel GAM webpage. Default: disabled
`mass1` - mass of the first body. Default value: 2
`mass2` - mass of the second body. Default value: 2
`spring1` - stiffness constant of the first spring. Default value: 5
`spring2` - stiffness constant of the second spring. Default value: 5
`damper1` - damping coefficient of the first damper. Default value: 3
`damper2` - damping coefficient of the second damper. Default value: 3
`saturationEnabled` - set equal to 1 to enable force saturation. Default: disabled
`maxForce` - maximum force allowed if force saturation is enabled. Default value: 100
`minForce` - minimum force allowed if force saturation is enabled. Default value: 0
`initialExtension1` - initial extension of the first spring. Default value: 0
`initialExtension2` - initial extension of the second spring. Default value: 0
`initialVelocity1` - initial extension rate of the first spring. Default value: 0
`initialVelocity2` - initial extension rate of the second spring. Default value: 0
`mass1Width` - width of the first body. Used in calculating the body positions and the animation. Default value: 1
`mass2Width` - width of the second body. Used in the animation. Default value: 1
`spring1Length` - rest length of the first spring. Default value: 2
`spring2Length` - rest length of the second spring. Default value: 2
`restitution1` - coefficient of restitution between wall and the first body. Default value: 1
`restitution2` - coefficient of restitution between the two bodies. Default value: 1

A.3 Plant Parameters

Free System

```
mass1 = 100.0
mass2 = 50.0
spring1 = 100.0
spring2 = 30.0
damper1 = 10.0
```

```
damper2 = 5.0
maxForce = 100.0
minForce = 0.0
initialExtension1 = -3.0
initialExtension2 = 20.0
initialVelocity1 = 0.0
initialVelocity2 = 0.0
mass1Width = 1.0
mass2Width = 0.5
spring1Length = 3.0
spring2Length = 5.0
saturationEnabled = 1
restitution1 = 0 (inelastic) or 1 (elastic)
restitution2 = 0 (inelastic) or 1 (elastic)
```

PID Controlled System

```
mass1 = 10.0
mass2 = 5.0
spring1 = 15.0
spring2 = 5.0
damper1 = 3.0
damper2 = 3.0
maxForce = 100.0
minForce = 0.0
initialExtension1 = 0.0
initialExtension2 = 0.0
initialVelocity1 = 0.0
initialVelocity2 = 0.0
mass1Width = 0.5
mass2Width = 0.5
spring1Length = 2.0
spring2Length = 2.0
saturationEnabled = 1
restitution1 = 0.75
restitution2 = 0.75
```