[An Electrical Example](http://book.xogeny.com/behavior/equations/electrical/)

[Lotka-Volterra Systems](http://book.xogeny.com/behavior/equations/population/)



[http://book.xogeny.com/_static/images/TitleHeading.png](http://book.xogeny.com/)

Release: v0.3.6-1-gc28d7f1-Early Access

[Basic Equations](http://book.xogeny.com/behavior/equations/)

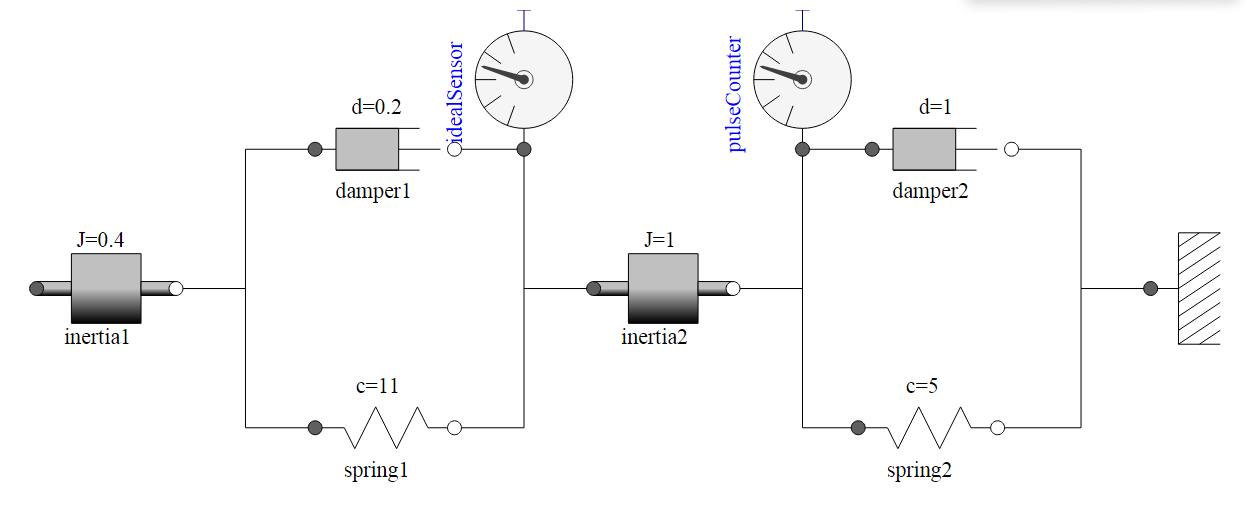
**A Mechanical Example**

Table Of Contents

* [Basic Equations](http://book.xogeny.com/behavior/equations/)
  + [A Mechanical Example](http://book.xogeny.com/behavior/equations/mechanical/)

A Mechanical Example

If you are more familiar with mechanical systems, this example might help reinforce some of the concepts we’ve covered so far. The system we wish to model is the one shown in the following figure:



It is worth pointing out how much easier it is to convey the intention of a model by presenting it in schematic form. Assuming appropriate graphical representations are used, experts can very quickly understand the composition of the system and develop an intuition about how it is likely to behave. While we are currently focusing on equations and variables, we will eventually work our way up to an approach (in the upcoming section of the book on[*Components*](http://book.xogeny.com/components/components/#components)) where **models will be built in this schematic form from the beginning**.

For now, however, we will focus on how to express the equations associated with this simple mechanical system. Each inertia has a rotational position, φ, and a rotational speed, ω where ω=φ˙. For each inertia, the balance of angular momentum for the inertia can be expressed as:

Jω˙=∑iτi

In other words, the sum of the torques, τ, applied to the inertia should be equal to the product of the moment of inertia, J, and the angular acceleration, ω˙.

At this point, all we are missing are the torque values, τi. From the previous figure, we can see that there are two springs and two dampers. For the springs, we can use Hooke’s law to express the relationship between torque and angular displacement as follows:

τ=kΔφ

For each damper, we express the relationship between torque and relative angular velocity as:

τ=dΔφ˙

If we pull together all of these relations, we get the following system of equations:

ω1J1ω˙1ω2J2ω˙2=φ˙1=k1(φ2−φ1)+d1d(φ2−φ1)dt=φ˙2=k1(φ1−φ2)+d1d(φ1−φ2)dt−k2φ2−d2φ˙2

Let’s assume our system has the following initial conditions as well:

φ1ω1φ2ω2=0=0=1=0

These initial conditions essentially mean that the system starts in a state where neither inertia is actually moving (*i.e.*, ω=0), but there is a non-zero deflection across both springs.

Pulling all of these variables and equations together, we can express this problem in Modelica as follows:

**model** **SecondOrderSystem** "A second order rotational system"

**type** **Angle**=Real(unit="rad");

**type** **AngularVelocity**=Real(unit="rad/s");

**type** **Inertia**=Real(unit="kg.m2");

**type** **Stiffness**=Real(unit="N.m/rad");

**type** **Damping**=Real(unit="N.m.s/rad");

**parameter** Inertia J1=0.4 "Moment of inertia for inertia 1";

**parameter** Inertia J2=1.0 "Moment of inertia for inertia 2";

**parameter** Stiffness k1=11 "Spring constant for spring 1";

**parameter** Stiffness k2=5 "Spring constant for spring 2";

**parameter** Damping d1=0.2 "Damping for damper 1";

**parameter** Damping d2=1.0 "Damping for damper 2";

Angle phi1 "Angle for inertia 1";

Angle phi2 "Angle for inertia 2";

AngularVelocity omega1 "Velocity of inertia 1";

AngularVelocity omega2 "Velocity of inertia 2";

**initial equation**

phi1 = 0;

phi2 = 1;

omega1 = 0;

omega2 = 0;

**equation**

*// Equations for inertia 1*

omega1 = der(phi1);

J1\*der(omega1) = k1\*(phi2-phi1)+d1\*der(phi2-phi1);

*// Equations for inertia 2*

omega2 = der(phi2);

J2\*der(omega2) = k1\*(phi1-phi2)+d1\*der(phi1-phi2)-k2\*phi2-d2\*der(phi2);

**end** **SecondOrderSystem**;

As we did with the low-pass filter example, RLC1, let’s walk through this line by line.

As usual, we start with the name of the model:

**model** **SecondOrderSystem** "A second order rotational system"

Next, we introduce physical types for a rotational mechanical system, namely:

**type** **Angle**=Real(unit="rad");

**type** **AngularVelocity**=Real(unit="rad/s");

**type** **Inertia**=Real(unit="kg.m2");

**type** **Stiffness**=Real(unit="N.m/rad");

**type** **Damping**=Real(unit="N.m.s/rad");

Then we define the various parameters used to represent the different physical characteristics of our system:

**parameter** Inertia J1=0.4 "Moment of inertia for inertia 1";

**parameter** Inertia J2=1.0 "Moment of inertia for inertia 2";

**parameter** Stiffness k1=11 "Spring constant for spring 1";

**parameter** Stiffness k2=5 "Spring constant for spring 2";

**parameter** Damping d1=0.2 "Damping for damper 1";

**parameter** Damping d2=1.0 "Damping for damper 2";

For this system, there are four non-parameter variables. These are defined as follows:

Angle phi1 "Angle for inertia 1";

Angle phi2 "Angle for inertia 2";

AngularVelocity omega1 "Velocity of inertia 1";

AngularVelocity omega2 "Velocity of inertia 2";

The initial conditions (which we will revisit shortly) are then defined with:

**initial equation**

phi1 = 0;

phi2 = 1;

omega1 = 0;

omega2 = 0;

Then come the equations describing the dynamic response of our system:

**equation**

*// Equations for inertia 1*

omega1 = der(phi1);

J1\*der(omega1) = k1\*(phi2-phi1)+d1\*der(phi2-phi1);

*// Equations for inertia 2*

omega2 = der(phi2);

J2\*der(omega2) = k1\*(phi1-phi2)+d1\*der(phi1-phi2)-k2\*phi2-d2\*der(phi2);

And finally, we have the closing of our model definition.

**end** **SecondOrderSystem**;

The only drawback of this model is that all of our initial conditions have been “hard-coded” into the model. This means that we will be unable to specify any alternative set of initial conditions for this model. We can overcome this issue, as we did with our [*Newton cooling examples*](http://book.xogeny.com/behavior/equations/physical/#getting-physical), by defining parametervariables to represent the initial conditions as follows:

**model** **SecondOrderSystemInitParams**

"A second order rotational system with initialization parameters"

**type** **Angle**=Real(unit="rad");

**type** **AngularVelocity**=Real(unit="rad/s");

**type** **Inertia**=Real(unit="kg.m2");

**type** **Stiffness**=Real(unit="N.m/rad");

**type** **Damping**=Real(unit="N.m.s/rad");

**parameter** Angle phi1\_init = 0;

**parameter** Angle phi2\_init = 1;

**parameter** AngularVelocity omega1\_init = 0;

**parameter** AngularVelocity omega2\_init = 0;

**parameter** Inertia J1=0.4 "Moment of inertia for inertia 1";

**parameter** Inertia J2=1.0 "Moment of inertia for inertia 2";

**parameter** Stiffness k1=11 "Spring constant for spring 1";

**parameter** Stiffness k2=5 "Spring constant for spring 2";

**parameter** Damping d1=0.2 "Damping for damper 1";

**parameter** Damping d2=1.0 "Damping for damper 2";

Angle phi1 "Angle for inertia 1";

Angle phi2 "Angle for inertia 2";

AngularVelocity omega1 "Velocity of inertia 1";

AngularVelocity omega2 "Velocity of inertia 2";

**initial equation**

phi1 = phi1\_init;

phi2 = phi2\_init;

omega1 = omega1\_init;

omega2 = omega2\_init;

**equation**

omega1 = der(phi1);

omega2 = der(phi2);

J1\*der(omega1) = k1\*(phi2-phi1)+d1\*der(phi2-phi1);

J2\*der(omega2) = k1\*(phi1-phi2)+d1\*der(phi1-phi2)-k2\*phi2-d2\*der(phi2);

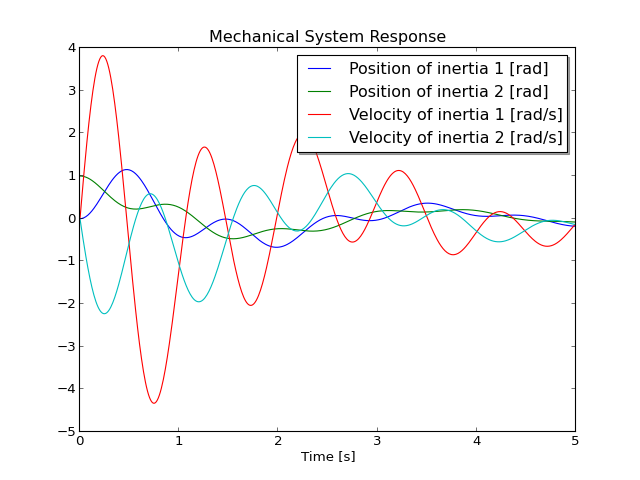
**end** **SecondOrderSystemInitParams**;

In this way, the parameter values can be changed either in the simulation environment (where parameters are typically editable by the user) or, as we will see shortly, via so-called “modifications”.

You will see in this latest version of the model that the values for the newly introduced parameters are the same as the hard-coded values used earlier. As a result, the default initial conditions will be exactly the same as they were before. But now, we have the freedom to explore other initial conditions as well. For example, if we simulate the SecondOrderSystemInitParams model as is, we get the following solution for the angular positions and velocities:

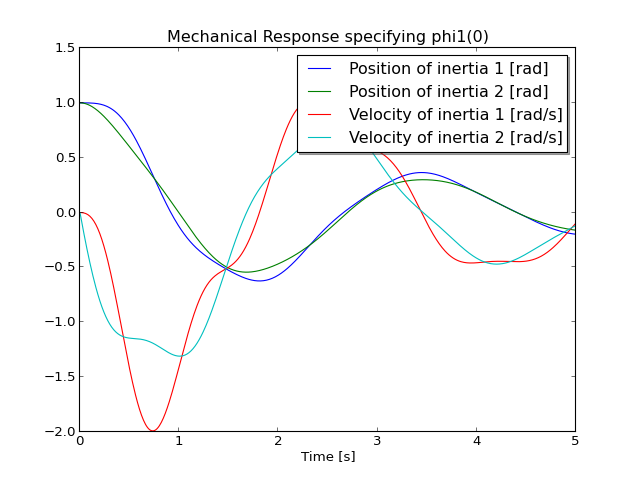
([Source code](http://book.xogeny.com/plots/SOSIP.py))

 Simulate SecondOrderSystemInitParams in your browser



However, if modify the phi1\_init parameter to be *1* at the start of our simulation, we get this solution instead:

([Source code](http://book.xogeny.com/plots/SOSIP1.py))



**Expanding on this mechanical example**

If you would like to see this example further developed, you may wish to jump to the set of examples involving rotational systems found in the section on[*Speed Measurement*](http://book.xogeny.com/behavior/discrete/measuring/#speed-measurement).

Otherwise, you can continue to the next set of examples which involve population dynamics.