MathModelica® System Designer Professional

A Sample Optimization Problem

Abstract

This notebook is an example of how the powerful scripting language of *Mathematica* can be utilized to solve non-trivial optimization problems that contain dynamic simulations. First we will define a Modelica model of a linear actuator with spring damped stopping and then a first order system. Using *MathModelica System Designer* scripting we will then find a damping for the translational spring-damper such that the step response is as "close" as possible to the step response from the first order system.

Initialization

Loading packages

To be able to use *MathModelica System Designer* within *Mathematica* we need to load the *MathModelica* package.

```
Needs["MathModelica`"]
```

Example

Consider the following model of a linear actuator with spring damped stopping:

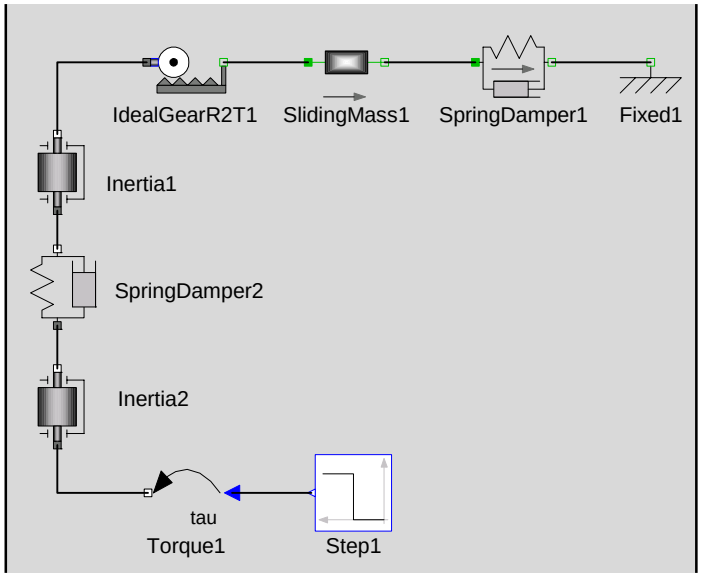


Diagram : LinearActuator

Below is the corresponding model code:

```
model LinearActuator
  Modelica.Blocks.Sources.Step step1;
  Modelica.Mechanics.Rotational.Torque torque1;
  Modelica.Mechanics.Rotational.Inertia inertia1(J=0.1);
  Modelica.Mechanics.Rotational.Inertia inertia2(J=0.1);
  Modelica.Mechanics.Rotational.SpringDamper springDamper2(c=15,d=2);
  Modelica.Mechanics.Rotational.IdealGearR2T idealGearR2T1;
  Modelica.Mechanics.Translational.SlidingMass slidingMass1(m=0.5);
  Modelica.Mechanics.Translational.SpringDamper springDamper1(c=20,d=3);
  Modelica.Mechanics.Translational.Fixed fixed1;

equation
  connect(springDamper1.flange_b,fixed1.flange_b);
  connect(slidingMass1.flange_b,springDamper1.flange_a);
  connect(idealGearR2T1.flange_b,slidingMass1.flange_a);
  connect(inertia2.flange_b,idealGearR2T1.flange_a);
  connect(springDamper2.flange_b,inertia2.flange_a);
  connect(inertia1.flange_b,springDamper2.flange_a);
  connect(torque1.flange_b,inertia1.flange_a);
  connect(step1.y,torque1.tau);
end LinearActuator;
```

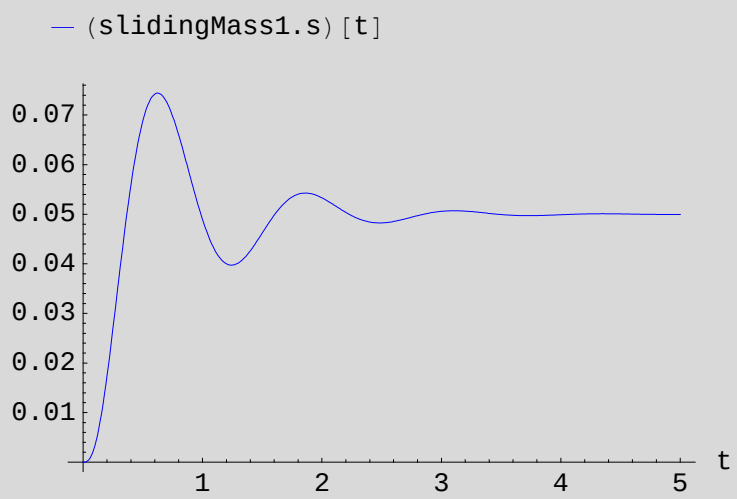
This model is available from the GettingStarted.Professional package delivered with *MathModelica System Designer* and we will use it in the rest of this example.

3.1 Optimization

The model can be simulated directly in *Mathematica* using the Simulate command of *MathModelica System Designer Professional*. We simulate a step response and store the result in the variable res0:

```
Simulate[GettingStarted.Professional.LinearActuator,
  {t, 0, 5}, ParameterValues → {springDamper1.d == 2}];
```

```
PlotSimulation[slidingMass1.s[t], {t, 0, 5}];
```

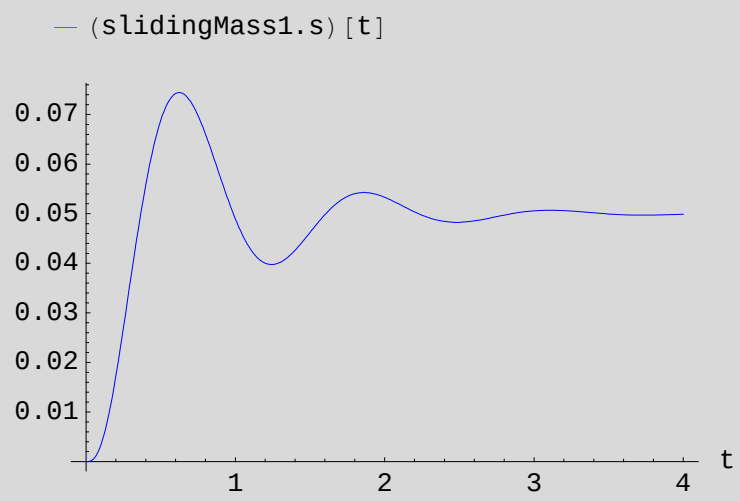


Suppose that we have some freedom in choosing the damping in the translational spring-damper. A number of simulation runs shows what kind of behavior we have for different values of the damping d. Table[ ] is used in Simulate[ ]to make simulations for damping 2 to 14, with a step of 2, ie seven simulations are performed.

```
res = Table[Simulate[GettingStarted.Professional.LinearActuator,
    {t, 0, 4}, ParameterValues → {springDamper1.d == q}], {q, 2, 14, 2}];
```

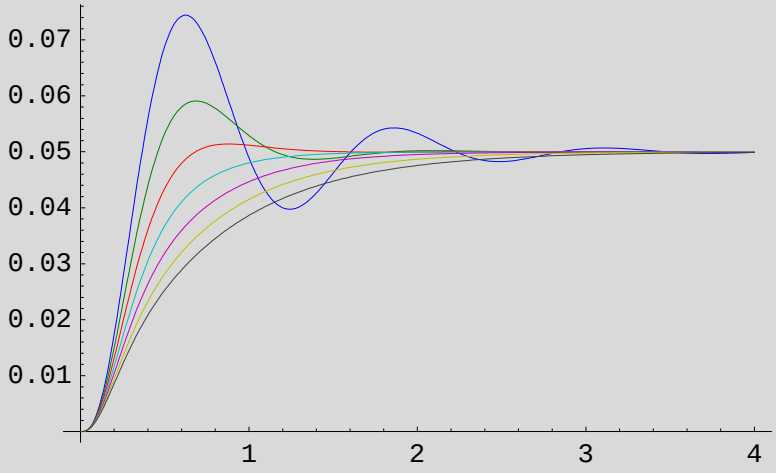
The simulation plot shows that the first simulation result from this batch the same as the result we obtained previously, which is to be expected as all parameters are the same for these two simulations.

```
PlotSimulation[slidingMass1.s[t],
  {t, 0, 4}, SimulationResult → res[[1]]];
```



It is more intersting to show how the behavior differs depending on the damping.

```
PlotSimulation[slidingMass1.s[t],
  {t, 0, 4}, SimulationResult → res, Legend → False];
```



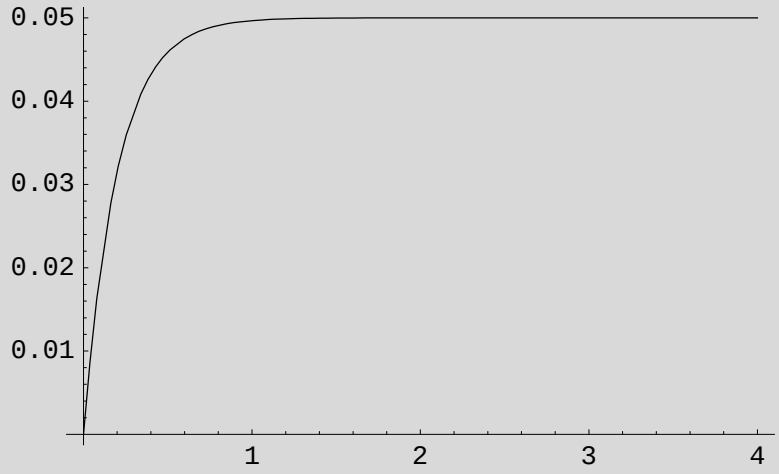
Now we want to get an optimal design for our system. We make sure that the variable *y* is not present in the global context in *Mathematica:*

```
Clear[y]
```

Now assume that we would like to choose the damping d so that the resulting system behaves as close as possible to the following first order system response:

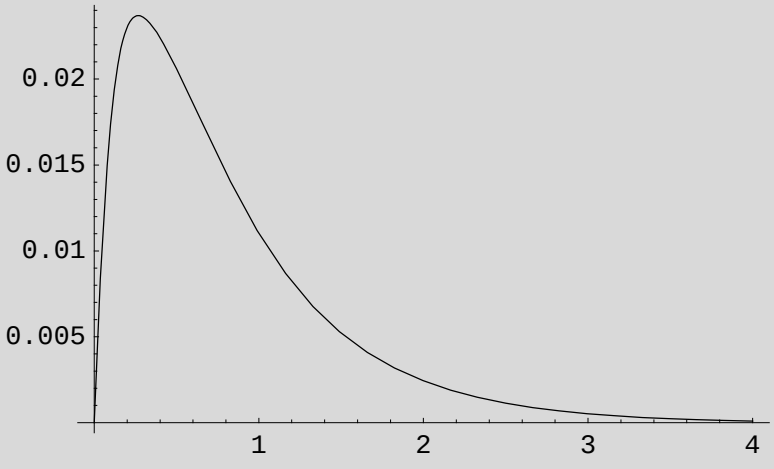
```
res1 = NDSolve[{0.2 y'[t] + y[t] == 0.05, y[0] == 0}, {y}, {t, 0, 4}];
```

```
Plot[y[t] /. res1, {t, 0, 4}, PlotRange -> All];
```



More interesting perhaps is to look at the difference between the desired signal and the step response.

```
Plot[(y[t] /. res1) - slidingMass1.s[t] /. ToInterpolationRules[
    res[[1]], slidingMass1.s], {t, 0, 4}, PlotRange → All];
```



Now, let us make things a little bit more automatic. Simulate and compute the integral of the square error from $t = 0$ to $t = 4$.

```
res = Simulate[GettingStarted.Professional.LinearActuator,
   {t, 0, 4}, ParameterValues → {springDamper1.d == 3}];
```

```
NIntegrate[First[(y[t] /. res1) - slidingMass1.s[t]]^2, {t, 0, 4}]

0.000162613
```

We define a function, f(a), doing the same thing as above, but for different spring-damper parameters $d = a$.

```
f[a_] := Module[{res, t},
  res = Simulate[GettingStarted.Professional.LinearActuator,
    {t, 0, 4}, ParameterValues → {springDamper1.d == a}];
  NIntegrate[First[(y[t] /. res1) - slidingMass1.s[t]]^2, {t, 0, 4}]]
```

and tabulate some results for $2 \le d = a \le 10$.

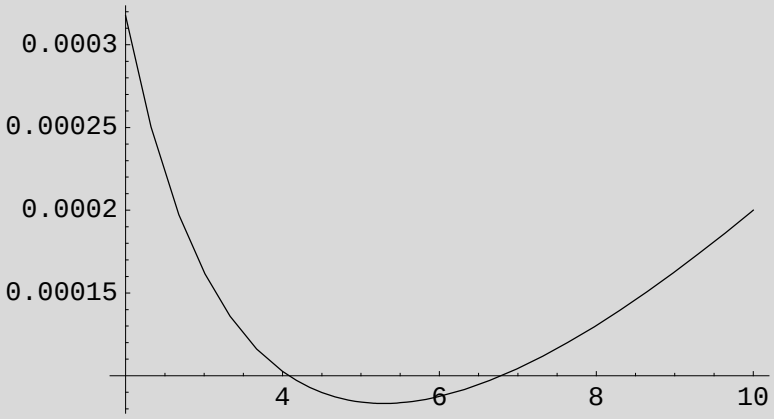
```
Table[{a, f[a]}, {a, 2, 10, .5}]

{{2, 0.00031762}, {2.5, 0.00022172},
 {3., 0.000162613}, {3.5, 0.000125287}, {4., 0.000102663},
 {4.5, 0.000089849}, {5., 0.0000840658}, {5.5, 0.0000836747},
 {6., 0.0000874684}, {6.5, 0.0000945919}, {7., 0.000104408},
 {7.5, 0.00011647}, {8., 0.000130405}, {8.5, 0.00014593},
 {9., 0.000162815}, {9.5, 0.000180902}, {10., 0.000200005}}
```

The tabulated values are interpolated using an interpolating function object. The default interpolation order is 3.

```
f_pre = Interpolation[%];
```

```
Plot[f_pre[a], {a, 2, 10}];
```



The minimizing value of *a* can be computed using `FindMinimum`:

```
FindMinimum[f_pre[s], {s, 5}]

{0.0000832595, {s → 5.28592}}
```

Now let us simulate with the optimal parameter value

```
Simulate[GettingStarted.Professional.LinearActuator,
  {t, 0, 4}, ParameterValues → {springDamper1.d == 5.2861}];
```

A plot comparing the first order system and linear actuator model response together with a plot of the squared error amplified with a factor 100.

```
Plot[{slidingMass1.s[t], y[t] /. res1,
   100 (slidingMass1.s[t] - (y[t] /. res1))^2}, {t, 0, 4}];
```