MathModelica® System Designer Professional

Hello World

1 Abstract

The most basic Modelica model is a differential equation. In this example a differential equation is implemented and simulated. Note that the Getting Started document shows how to develop the same model in the Model Editor and study plots in the Simulation Center.

2 Initialization

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

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

3 Model

There is a long tradition that the first example in any computer language is a trivial program printing the string "Hello World"?. Since Modelica, the language used in MathModelica, is an equation-based language, printing a string does not make much sense. Instead our Hello World Modelica program solves a trivial differential equation:

$\dot{x} = -a\,x$

The variable x in this equation is a dynamic variable (here also a state variable) that can change value over time. The time derivative is the derivative of x, represented as der(x) in Modelica.

All Modelica programs consist of class declaration (class, block, model, package, etc) and in this case we declare the program as a model. Note that when working with Modelica models in *Mathematica* it is recommended that you use the MathModelica stylesheet (this can be found in the Format>Style Sheet menu) as this contain a special input cell called ModelicaInput. Even though you can type you Modelica models in normal Input cells, ModelicaInput cells are better suited for these purpose.

Just as any other *Mathematica* input ModelicaInput cells are evaluated by pressing shift+enter.

```
model HelloWorld
Real x(start=1);

equation
  der(x)=-x;
end HelloWorld;
```

4 Simulation

The *MathModelica System Designer Professional* command **Simulate** simulates the model given by the *Modelica* code given above.

```
?? Simulate
```

```
Simulate[model,{t,tmin,tmax}]
  simulates the model in the range tmin to tmax.
```

```
Attributes[Simulate] = {ReadProtected}

Options[Simulate] = {Method → Dassl, Tolerance → 0.0001,
  IntervalLength → 0, NumberOfIntervals → 500, InitialValues → {},
  ParameterValues → {}, InteractiveVariables → True}
```

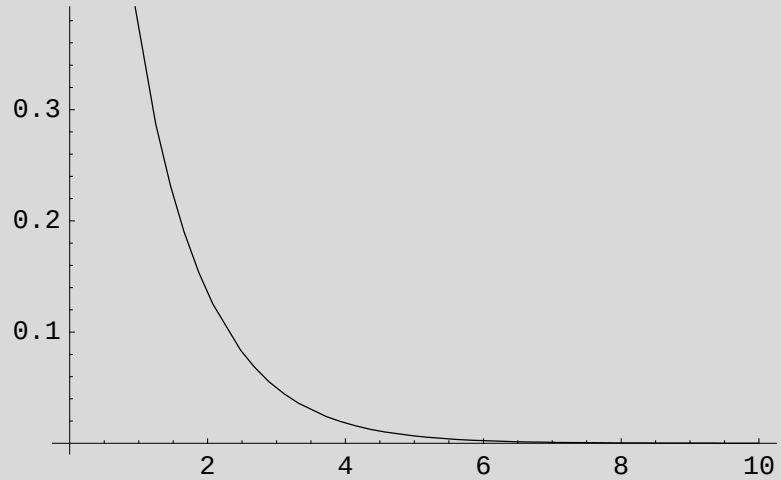
We simulate the HelloWorld example.

```
Simulate[HelloWorld, {t, 0, 10}]

<SimulationData:  :  : {0., 10.} :
  504 data points : 1 events : 2 variables>
{x, x′}
```

The Simulate command returns a SimulationData object containing all model parameters and variables in the form of interpolating functions. It is possible to use the *Mathematica* Plot command to plot simulation results.

```
Plot[x[t], {t, 0, 10}];
```

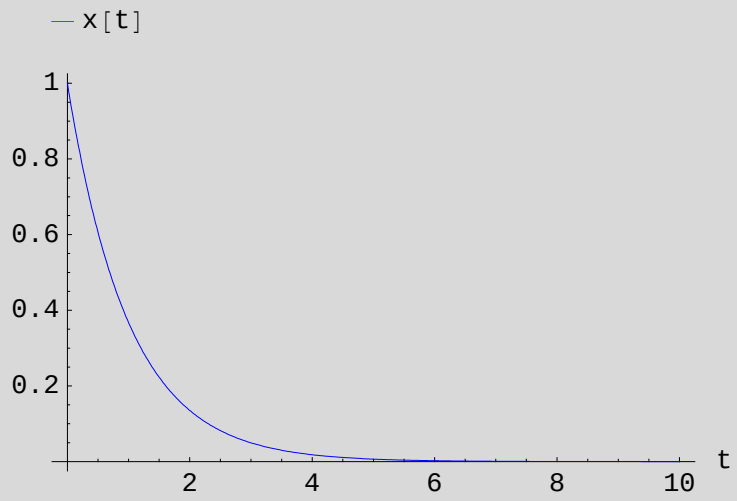


However, it is better to use the PlotSimulation command as this is especially adapted to handle simulation results, including the possibility to plot simulation results from several different simulations, using the SimulationResult option, in the same plot, as well as the possibility to plot hierarchical variables, which are very often used in modeling.

```
?PlotSimulation
```

```
PlotSimulation[f,{t,tmin,tmax}] generates a plot
  of the signal f as a function of t from tmin to tmax.
  PlotSimulation[{f1,f2,...},{t,tmin,tmax}] plots several
  functions fi. PlotSimulation[{"var1","var2"},{t,tmin,tmax}]
  plots variables having the Modelica names var1 and var2.
  PlotSimulation[f,{t,tmin,tmax},SimulationResult->res]
  generates a plot of f using simulation result res. Default
  value of res is the result from the latest simulation.
```

```
PlotSimulation[x[t], {t, 0, 10}];
```



In many cases you want to compare results for different simulations, for instance you might want to test different initial or parameter values.

```
?? Simulate
```

```
Simulate[model,{t,tmin,tmax}]
  simulates the model in the range tmin to tmax.
```

```
Attributes[Simulate] = {ReadProtected}

Options[Simulate] = {Method → Dassl, Tolerance → 0.0001,
  IntervalLength → 0, NumberOfIntervals → 500, InitialValues → {},
  ParameterValues → {}, InteractiveVariables → True}
```

We can make several simulations and store the results in different variables.

```
res1 = Simulate[HelloWorld, {t, 0, 10}];
res2 = Simulate[HelloWorld, {t, 0, 10}, InitialValues → {x == -10}];
res3 = Simulate[HelloWorld, {t, 0, 10}, InitialValues → {x == 10}];
```

Then we can use the SimulationResult option to compare the results in one plot.

```
PlotSimulation[x[t], {t, 0, 10}, SimulationResult → {res1, res2, res3}];
```