Implementation of Comsol in Simulink S-Functions, Revisited

A.W.M. (Jos) van Schijndel
Eindhoven University of Technology, A.W.M.v.Schijndel@tue.nl;

Abstract: Comsol has standard facilities to export models to SimuLink. Normally, the standard export works well if the solvers, available in SimuLink, can handle the problem. However, if a model in Comsol needs special solvers, for example airflow or other non-linear problems, the standard export to SimuLink is often not suitable, because the standard solvers of SimuLink cannot handle such a problem efficiently. This can cause long simulation duration times and even leads to no solution at all. The paper presents a possible solution to this problem by implementing Comsol code in the discrete section of a SimuLink S-Function The advantage of this approach is that the special solvers of Comsol can be used in the SimuLink environment. This can lead to significant improvement of the simulation duration time.

Keywords: S-Function, SimuLink, Comsol, non linear, implementation

1. Introduction

Comsol has standard facilities to export models to SimuLink. Normally, the standard export works well if the solvers, available in SimuLink, can handle the problem. However, if a model in Comsol needs special solvers, for example airflow or other non-linear problems, the standard export to SimuLink is often not suitable, because the standard solvers of SimuLink cannot handle such a problem efficiently. This can cause long simulation duration times and even leads to no solution at all.

Similar earlier work, based on FemLab (former Comsol), was already published in 2005. Due to the quite different MatLab interface in Comsol compared with FemLab, it is worthwhile to update this past work to the most recent Comsol version.

Section 2 introduces the concept of S-Functions in SimuLink.

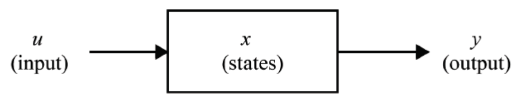
Section 3 presents the approach by implementing Comsol code in the discrete section of a SimuLink S-Function. The S-Function solves each time step the non-linear problem using the Comsol solver. After each time step the solution is exported. Furthermore, different boundary values can be applied at will. A simple heating problem is used to show the approach.

Section 4 provides a former case study using Comsol, where the airflow temperature at the inlet of an office is controlled by an on/off controller in SimuLink.

Section 5 presents the conclusions.

2. S-Functions in SimuLink

The main idea is to use S-Functions (Mathworks 1998). In order to have a good understanding of how S-Functions work, a short summary is provided here. An S-function (system-function) is a computer language description of a dynamic system available in SimuLink. The form of an S-function is very general and can accommodate continuous, discrete and hybrid systems. As a matter of fact, nearly all SimuLink models can be described as S-functions. Each block within a SimuLink model has the following general characteristics: a vector of inputs, u, a vector of outputs, y, and a vector of states, x, as shown by the illustration in Figure 1.



$$y = f_0(t, x, u) \quad \text{(output)}$$

$$\dot{x}_c = f_d(t, x, u) \quad \text{(derivative)}$$

$$x_{d_{k+1}} = f_u(t, x, u) \quad \text{(update)}$$

$$\text{where} \quad x = x_c + x_d$$

Figure 1. Illustration of the general characteristics of an S-Function

The state vector may consist of continuous states, discrete states, or a combination of both. The mathematical relationships between the inputs, outputs, and the states are expressed by the equations presented in the above figure. SimuLink partitions the state vector x into two parts: the continuous states xc and the discrete states xd. The continuous states occupy the first part of the state vector, and the discrete states occupy the second part. For blocks with no states, x is an empty vector. SimuLink makes repeated calls during specific stages of the simulation to each block in the model, directing it to perform tasks such as computing its outputs, updating its discrete states, or computing its derivatives. Additional calls are made at the beginning and end of a simulation to perform initialization and termination tasks. Figure 2 illustrates how SimuLink performs a simulation.

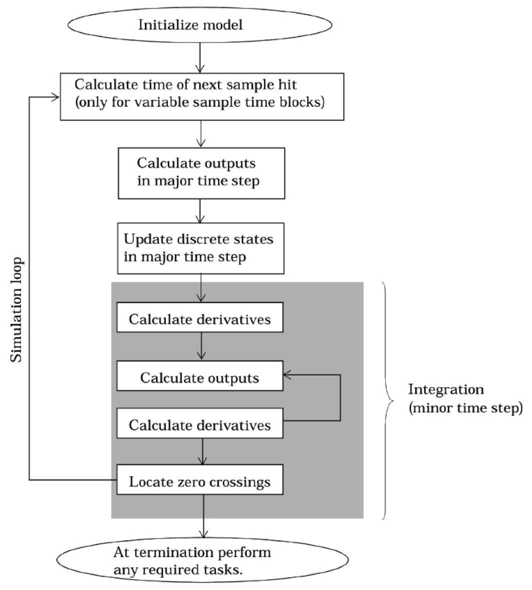

Initialize model
Calculate time of next sample hit (only for variable sample time blocks)
Calculate outputs in major time step
Update discrete states in major time step
Calculate derivatives
Calculate outputs
Calculate derivatives
Locate zero crossings
Simulation loop
Integration (minor time step)
At termination perform any required tasks.

Figure 2. How SimuLink performs simulation

First, SimuLink initializes the model; this includes initializing each block, including S-functions. Then SimuLink enters the simulation loop, where each pass through the loop is referred to as a simulation step. During each simulation step, SimuLink executes the S-function block. This continues until the simulation is complete (see Figure 2). SimuLink makes repeated calls to S-functions in a model. During these calls, SimuLink calls S-function routines, which perform tasks required at each stage. These tasks include: Initialization, calculation of next sample hit, calculation of outputs in the major time step, update discrete states in the major time step, and integration.

3. Approach

3.1 A dummy S-Function

The first step is to develop a 'dummy' S-Function, with a fixed time step. Figure 3 shows the SimuLink model.

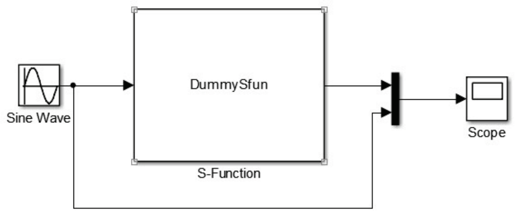


Figure 3. The dummy S-Function

The mfile i.e. the contents of the S-Function is presented in Appendix 1. This dummy model imports the time step from the SimuLink GUI and for each time step it reads the input (the Sine Wave) performs a dummy Comsol simulation at the mdlUpdate function and produces dummy output at the mdlOutputs function.

3.2 A simple heating model in Comsol

A very simple 2D Comsol model was developed as demonstration model later on to be implemented into the S-Function of SimuLink. Figure 4 shows the geometry.

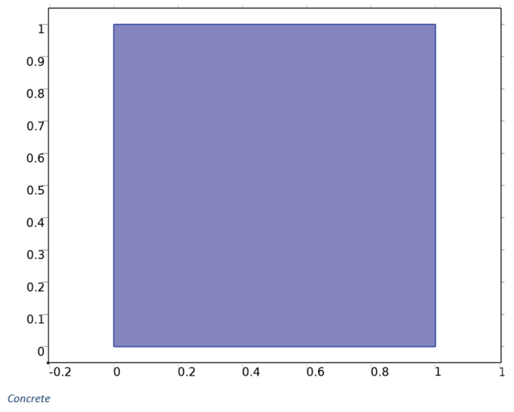


Figure 4. The geometry of the simple heating model

Other modeling selected features are (1) Physics: Heat Transfer in Solids; (2) Heat Source: 1000

W/m^3 (3) Materials: Concrete (4); Study: Time Dependent with range (0,3600,24*3600).

The result is a uniform heating of the concrete square from 293.15 K (the default initial value) to 336 K. The Comsol model was exported to a MatLab mfile. This is provided in Appendix 3.

3.3 Additional manual settings

In this Section additional manual setting are provided. To simulate again and taking the effect of a parameter change into account, the user can type the following command:

```
Cmodel.sol('sol1').runAll;
```

This forces a new simulation run.

3.3.1 Input

The input parameter (u) is the heat source. This parameter can be set manually by:

```
Cmodel.physics('ht').feature('hs1').
set('Q', 1, '500');
```

This changes the heat source from 1000 to 500 W/m^3

3.3.2 Time range

The time range is set by

```
Cmodel.sol('sol1').feature('t1').set
('tlist',
'range(0,3600,2*24*3600)');
```

Here the end time is changed to 48 hours.

3.3.3 Init values

The init values are set by:

```
Cmodel.physics('ht').feature('init1'
).set('T', 1, '283.15[K]');
```

The initial temperature is changed to 283.15 K

3.3.4 Output

The output (y) is the temperature at the centre of the square, evaluated for the last time step. This parameter can be set manually by:

```
y=mphinterp(Cmodel,'T','coord',[0.5;
0.5],'Solnum','end')
```

The last step for the final implementation is to run 24 steps with a range of 0 to 3600 sec and update the initial values with each time step. The mfile is shown in Appendix 2. The result is presented in Figure 5.

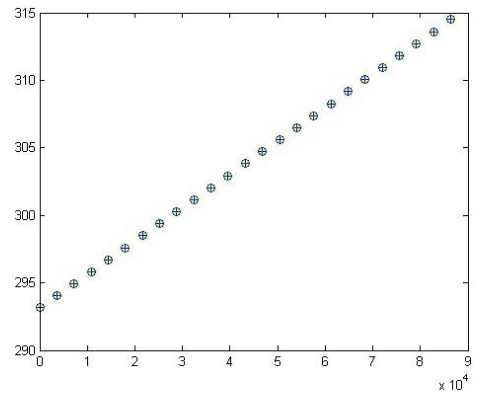


Figure 5. Comparison with a simulation of the whole period at once (+) and manually stepping (o)

The latter shows how to split a complete simulation into several time steps using MatLab commands. Using the results of the previous Sections we are now able to combine all mfiles for the final implementation.

3.4 Final implementation

The final part is to copy and paste the mfiles of Appendix 2 & 3 into the S-Function of Appendix 1. The reader should notice that model of appendix 3 should be copied into the S-Function of Appendix 4 (at the right place see comment) in order to get a proper S-Function. To verify the working of S-Function with the included Comsol (i.e. Appendix 4), the simple heating model was controlled with a Relay in SimuLink. Figure 6 shows the SimuLink model. The Relay controls the temperature between 288 K and 290 K by providing heat 1000 (W/m^3) or cool -1000 (W/m^3) to the model. In Figure 7 the simulation results are presented. The proper switching of the controller and response of the model is observed. This is a verification of the implementation of the work so far.

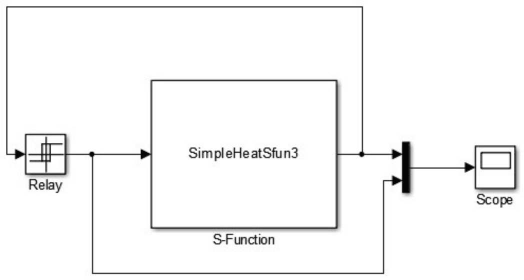


Figure 6. The simple heating model in SimuLink with Relay controller.

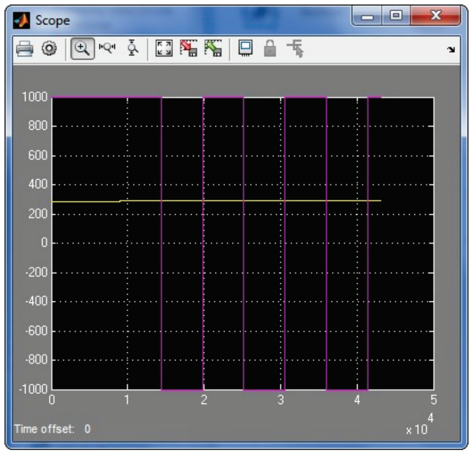


Figure 7. The simulated result

3.5 Discussion

At this point the reader should notice that the goal of this section was to demonstrate *how* Comsol models can be implemented into S-Functions. Therefore a very simple model and controller were used. The same results can be obtained by Comsol without the use of SimuLink. However there a least two main reasons why the use of S-Functions are favorable:

(1) If you have complicated controllers that cannot be modeled in Comsol (yet).

(2) If you have a Comsol model that will not run with the present standard export facility of Comsol to SimuLink.

5. Conclusion

It is concluded that Comsol models can also be exported to SimuLink by writing an appropriate S-Function. The advantage of this approach is that the special solvers of Comsol can be used in the SimuLink environment. This can lead to significant improvement of the simulation duration time. Furthermore it provides the possibility to use advanced controller facilities of SimuLink with Comsol models.

References

Schijndel, A.W.M. van (2014). A review of the application of SimuLink S-functions to multi domain modeling and building simulation. Journal of Building Performance Simulation, 7(3), 165-178.

Schijndel, A.W.M. van (2009). Integrated modeling using MatLab, SimuLink and COMSOL : with heat, air and moisture applications for building physics and systems. Saarbrücken: VDM Verlag Dr. Müller, XIV, 197 pp.

Schijndel, A.W.M. van (2005). Implementation of FemLab in S-Functions. Proceedings of FemLabConference FrankFurt, 2-4 November 2005. (pp. 324-329). Frankfurt.

Appendix 1

```
function [sys,x0,str,ts] = DummySfun(t,x,u,flag)
% Diskrete S-function Working almost empty for Comsol
comsolmodel=[];
%Init my variables
tstap=eval(get_param(gcs,'fixedstep'));  % time step in [s]
u0=0;     %
switch flag,
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(comsolmodel,tstap,u0);
  case 2,
    sys = mdlUpdate(t,x,u,comsolmodel,tstap);
  case 3,
    sys = mdlOutputs(t,x,u,comsolmodel,tstap);
  case 9,
   otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts] = mdlInitializeSizes(comsolmodel,tstap,u0)
nT=1; %Dummy replace with number of nodes if neccessary
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = nT;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = u0*zeros(nT,1);
str = [];
ts  = [tstap 0];

function sys = mdlUpdate(t,x,u,comsolmodel,tstap)
disp(['u =' num2str(u(1)) ])
disp(['calc ' num2str(t) ' to ' num2str(t+tstap) ])
sys=1; %Dummy replace with nT number of nodes if neccessary

function sys = mdlOutputs(t,x,u,comsolmodel,tstap)
sys=1; %Dummy replace with nT number of nodes if neccessary
```

Appendix 2

```
Cmodel.physics('ht').feature('init1').set('T', 1, '293.15[K]');
Tmid_all5(1)=293.15;
for i=1:24
Cmodel.sol('sol1').feature('t1').set('tlist', 'range(0,3600)');
Cmodel.sol('sol1').runAll;
u=mphinterp(Cmodel,'T','coord',[0.5;0.5],'Solnum','end')
Tmid_all5(i+1)= u;
ustr=[num2str(u) '[K]'];
Cmodel.physics('ht').feature('init1').set('T', 1, ustr);
end
```

Appendix 3

```
import com.comsol.model.*
import com.comsol.model.util.*
Cmodel = ModelUtil.create('Model');
Cmodel.modelPath('D:\04_Comsol44\SimuLink');
Cmodel.modelNode.create('comp1');
Cmodel.geom.create('geom1', 2);
Cmodel.mesh.create('mesh1', 'geom1');
Cmodel.physics.create('ht', 'HeatTransfer', 'geom1');
Cmodel.geom('geom1').run;
Cmodel.study.create('std1');
Cmodel.study('std1').feature.create('time', 'Transient');
Cmodel.study('std1').feature('time').activate('ht', true);
Cmodel.geom('geom1').feature.create('sq1', 'Square');
Cmodel.geom('geom1').run;
Cmodel.material.create('mat1');
Cmodel.material('mat1').name('Concrete');
Cmodel.material('mat1').set('family', 'concrete');
Cmodel.material('mat1').propertyGroup('def').set('density', '2300[kg/m^3]');
Cmodel.material('mat1').propertyGroup('def').set('thermalconductivity', '1.8[W/(m*K)]');
Cmodel.material('mat1').propertyGroup('def').set('heatcapacity', '880[J/(kg*K)]');
Cmodel.material('mat1').selection.geom('geom1', 2);
Cmodel.material('mat1').selection.set([1]);
Cmodel.material('mat1').set('family', 'concrete');
Cmodel.name('Simple2D_fase2.mph');
Cmodel.physics('ht').feature.create('hs1', 'HeatSource', 2);
Cmodel.physics('ht').feature('hs1').selection.set([1]);
Cmodel.physics('ht').feature('hs1').set('Q', 1, '1000');
Cmodel.study('std1').feature('time').set('tlist', 'range(0,3600,24*3600)');
Cmodel.sol.create('sol1');
Cmodel.sol('sol1').study('std1');
Cmodel.sol('sol1').feature.create('st1', 'StudyStep');
Cmodel.sol('sol1').feature('st1').set('study', 'std1');
Cmodel.sol('sol1').feature('st1').set('studystep', 'time');
Cmodel.sol('sol1').feature.create('v1', 'Variables');
Cmodel.sol('sol1').feature('v1').set('control', 'time');
Cmodel.shape('shape1').feature('shfun1');
Cmodel.sol('sol1').feature.create('t1', 'Time');
Cmodel.sol('sol1').feature('t1').set('tlist', 'range(0,3600,24*3600)');
Cmodel.sol('sol1').feature('t1').set('plot', 'off');
Cmodel.sol('sol1').feature('t1').set('plotfreq', 'tout');
Cmodel.sol('sol1').feature('t1').set('probesel', 'all');
Cmodel.sol('sol1').feature('t1').set('probes', {});
Cmodel.sol('sol1').feature('t1').set('probefreq', 'tsteps');
Cmodel.sol('sol1').feature('t1').set('atolglobalmethod', 'scaled');
Cmodel.sol('sol1').feature('t1').set('atolglobal', 0.0010);
Cmodel.sol('sol1').feature('t1').set('estrat', 'exclude');
Cmodel.sol('sol1').feature('t1').set('maxorder', 2);
Cmodel.sol('sol1').feature('t1').set('control', 'time');
Cmodel.sol('sol1').feature('t1').feature.create('fc1', 'FullyCoupled');
Cmodel.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
Cmodel.sol('sol1').feature('t1').feature('fc1').set('damp', 0.9);
Cmodel.sol('sol1').feature('t1').feature('fc1').set('maxiter', 5);
Cmodel.sol('sol1').feature('t1').feature.create('d1', 'Direct');
Cmodel.sol('sol1').feature('t1').feature('d1').set('linsolver', 'pardiso');
Cmodel.sol('sol1').feature('t1').feature('fc1').set('linsolver', 'd1');
Cmodel.sol('sol1').feature('t1').feature('fc1').set('jtech', 'once');
Cmodel.sol('sol1').feature('t1').feature('fc1').set('damp', 0.9);
Cmodel.sol('sol1').feature('t1').feature('fc1').set('maxiter', 5);
Cmodel.sol('sol1').feature('t1').feature.remove('fcDef');
Cmodel.sol('sol1').attach('std1');
```

Appendix 4

```
function [sys,x0,str,ts] = SimpleHeatSfun3(t,x,u,flag)
tstap=eval(get_param(gcs,'fixedstep'));  % time step in [s]
u0=283.15;
switch flag,
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(t,x,u,tstap,u0);
  case 2,
    sys = mdlUpdate(t,x,u,tstap);
  case 3,
    sys = mdlOutputs(t,x,u,u0);
  case 9,
   otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts] = mdlInitializeSizes(t,x,u,tstap,u0)
global Cmodel
% *** Copy the model of Appendix 3 Here  !!!!!**

Cmodel.physics('ht').feature('init1').set('T', 1, [num2str(u0) '[K]']);

nT=1; %Number of States
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = nT;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = u0*zeros(nT,1);
str = [];
ts  = [tstap 0];

function sys = mdlUpdate(t,x,u,tstap)
global Cmodel
disp(['u =' num2str(u(1)) ])
Cmodel.physics('ht').feature('hs1').set('Q', 1, num2str(u(1)));
disp(['calc ' num2str(t) ' to ' num2str(t+tstap) ])
Cmodel.sol('sol1').feature('t1').set('tlist', ['range(0,' num2str(tstap) ')']);
Cmodel.sol('sol1').runAll;
y=mphinterp(Cmodel,'T','coord',[0.5;0.5],'Solnum','end')
Cmodel.physics('ht').feature('init1').set('T', 1, [num2str(y) '[K]']);
sys=y;

function sys = mdlOutputs(t,x,u,u0)
global Cmodel
if t>0
y= mphinterp(Cmodel,'T','coord',[0.5;0.5],'Solnum','end');
else
y=u0;
end

sys=y;
```