WORKSHOP 2.2

Understanding equation-based modeling

Contents

Introduction	1
Creating the package	1
Boundary conditions and Initialization: Mechanical system	1
Mass in gravity field	1
Boundary condition 1	2
Boundary condition 2	4
Boundary condition 3	5
Initialization	б
Boundary conditions and Initialization: Thermodynamic system	7
Mass flow calculation	7
Boundary condition 1	8
Boundary condition 2	9
Boundary condition 3	9
Initialization	9
Identifying Degrees of Freedom10	0
Inverting system models1	1

Introduction

In this workshop, the concepts of equation-based and component-based modeling are investigated. The workshop contains four main parts: specifying boundary conditions, initialization, identifying degrees of freedom, and inverting system models. There are parallel exercises for boundary conditions and initialization, one track for mechanical systems and one track for thermodynamic system; they will both show the same features of Modelica.

Creating the package

1. Create a new package **W2**. If applicable, place this inside the course package you created for the previous lectures. For example, the complete path could be *TrainingPack.Day2.W2*.

Boundary conditions and Initialization: Mechanical system

Mass in gravity field

1. In W2, create a model called MassInGravityField.



- 2. Drag in a *Modelica.Mechanics.MultiBody.World*, note that this component will automatically be called **world**, and get the prefix *inner*.
- 3. Drag in a *Modelica.Mechanics.MultiBody.Parts.Body*, name it mass and set the mass to 1 kg, see the image below**Error! Reference source not found.**.



Note: Make sure you name it mass, otherwise modify the examples according to your naming convention.

4. Simulate using default settings (Dynamic, Stop Time = 1s).

Boundary condition 1

- 5. Extend **MassInGravityField** (right-click on model name in *Library Browser* and select *Extend...*) and call it **MassWithForce**.
- 6. Add a *Modelica.Mechanics.MultiBody.Forces.WorldForce* and call it force.
- 7. Add a *Modelica.Blocks.Sources.RealExpression*.
- 8. We now need to switch to the code layer to make some changes. Open the code editor by selecting "Code" in the dropdown at the top of the modeling canvas:

Diagram	-	[
Diagram		
Code	-	_

a. We need to make a size-3 array of this component so that we can provide an input for the force along each coordinate direction. To do this, append the string "[3]" to **realExpression**.

mod	el MassWithForce					-
	extends ImpactTrainingSolution	s.Day2.W2.Mechan	icalTrack.Mas	sInGravit	tyFiel	d;
	.Modelica.Mechanics.MultiBody.	Forces.WorldForc	e force annot	ation(•••);	
	.Modelica.Blocks.Sources.RealE	xpression realEx	<pre>(pression[3] a</pre>	nnotatior	n());
	annotation(••••);					
end	MassWithForce;					

b. Create a variable *Modelica*. Units. SI. Force f[3].

1	model MassWithForce
2	<pre>extends ImpactTrainingSolutions.Day2.W2.MechanicalTrack.MassInGravityField;</pre>
3	.Modelica.Mechanics.MultiBody.Forces.WorldForce force annotation();
4	<pre>.Modelica.Blocks.Sources.RealExpression realExpression[3] annotation(••••);</pre>
5	.Modelica.Units.SI.Force f[3];
6	annotation(····);
7	end MassWithForce;

- c. Use this force variable in the **realExpression** component, see the figure below.
 - 1 model MassWithForce
 2 extends ImpactTrainingSolutions.Day2.W2.MechanicalTrack.MassInGravityField;
 3 .Modelica.Mechanics.MultiBody.Forces.WorldForce force annotation(....);
 4 .Modelica.Blocks.Sources.RealExpression realExpression[3](y = f) annotation(....);
 5 .Modelica.Units.SI.Force f[3];
 6 annotation(....);
 7 end MassWithForce;
- d. Save the changes and close the code editor.
- 9. Create connections between **realExpression**, **force** and **mass**. Make sure to select ":" for the **realExpression** and **force** connectors.

realExpression [: • force	
⊳ y 🕞 🕨 force[: ♥]	

10. The model should now look like the screenshot below.



11. We need to make one additional change in the code layer. In the equation section, add an equation for the position of the mass: $mass.frame_a.r_0 = \{sin(time), 0.1*time, cos(time)\};$. The result should now look like the following figure.

1	model MassWithForce
2	extends .Workspace.Day2.Workshop2.MassInGravityField;
3	.Modelica.Mechanics.MultiBody.Forces.WorldForce force annotation(•••);
4	.Modelica.Blocks.Sources.RealExpression realExpression[3](y=f) annotation();
5	Modelica.SIunits.Force f[3];
6	equation
7	<pre>mass.frame_a.r_0 = {sin(time), 0.1*time, cos(time)};</pre>
8	<pre>connect(realExpression.y,force.force) annotation();</pre>
9	<pre>connect(force.frame_b,mass.frame_a) annotation();</pre>
10	end MassWithForce;

12. It is now time to simulate our model. To better visualize the results, we can use the 3D animation capability in Modelon Impact.

Simulate for 10 seconds and inspect the animation (right-click on canvas and select *View 3D animation*). What does this imply? Compare to the last exercise in this workshop.

Boundary condition 2

- 13. Extend MassInGravityField and call it MassInPointGravityField.
- 14. Set the initial position and velocity of the **mass** to $\{0,1,0\}$ and $\{1,0,0\}$, respectively. See figure below.

	Mass Rigid body with r Modelica.Mechar	nass, ir nics.Mu	ertia ten ItiBody.P	sor and one f Parts.Body	rame conne	ctor
INFORM	ATION					>
PROPER General	TIES Initialization	Anin	nation	Advanced	Variable	Y
r_0 star stat	t eSelect	÷	{0,1,0] alwa	} ays else Stat	m 🗹 eSelect.avo	m » id •
v_0 star stat	t eSelect	:	{1,0,0] alwa	} ays else Stat	m/s 🔽 eSelect.avo	m/s » id •

15. In the **world** component, select *point gravity field*, and set $mue = 1 \text{ m}^3/\text{s}^2$ as shown in the figure below.

У.	 WOrld World coordinate system + gravity field + default animation de Modelica.Mechanics.MultiBody.World 							
П	NFORMA	TION				>		
F	ROPERT	IES				~		
	General	Animation	Defau	lts		T		
	gravit	vAcceleration				~		
	enableA	nimation	:	✓	»			
	animate	World	:	✓	»			
	animate	Gravity	:	✓	»			
	animate	Ground	:		»			
	label1		:	"x"		-		
	label2		:	"y		-		
	gravityTy	/pe	:	Po	int gravity field	× -		
	g		:	M	odelica.Constants.g_n	m/s²		
2	n		:	{0,	-1,0} "negative y axis"	-		
	mue		:	1		m³/s²		
	driveTrai	nMechanics3D	•	✓	»			

16. Simulate for 10 seconds and inspect the 3D animation.

Boundary condition 3

- 17. Extend MassInGravityField and call it SuspendedMass.
- Add a *Modelica.Mechanics.MultiBody.Parts.Fixed* and a *Modelica.Mechanics.MultiBody.Forces.SpringDamperParallel* with 25 N/m stiffness and 1 Ns/m damping.
- 19. Set the initial position of the mass " $r_0.start$ " to {0,0.1,0}. Make sure to check the checkbox to make the start value fixed.

		MASS Rigid body with n Modelica.Mechan	ass, inertia tensor and one frame connecto ics.MultiBody.Parts.Body
I	NFORMA	TION	>
Ρ	ROPERT	IES	~
(General	Initialization	Animation Advanced Variables
	r_0		: m]
	start state	Select	{0,0.1,0} m v » always else StateSelect.avoid v

20. Also set the states in the mass, on the Advanced tab, check enforceStates:

General	Initialization	Animation	Advanced	Variables	
• enforceSt	ates	:	🗹 »		
• useQuate	rnions	:	✓ »		
sequence	_angleStates	:	{1,2,3} "Card	lan/Tait-Bryan angles"	-

21. Connect the **springDamper** to the **mass** and to the **fixed** to get a model like the one below.





22. Simulate for 10 seconds and inspect the 3D animation.

Initialization

- 23. Extend SuspendedMass and call it SuspendedMassSteadyState.
- 24. In the parameter dialog of **mass**, fix the initial conditions of velocity and acceleration at zero. (Also set angles and w_0_start =0 and w_0_fixed = true, to enforce the rotational states and uncheck fixed for r_0 .)

General Initialization	Animation Advanced Variables
r_0	: m
v_0	: m/s
start stateSelect	{0,0,0} m/s always else StateSelect.avoid
a_0	: [m/s²]
start	{0,0,0} m/s² ♥ »
 angles_fixed 	: 🔽 »
angles_start	: 0 0 0 deg »
 sequence_start 	: {1,2,3} "Cardan/Tait-Bryan an 💌
 w_0_fixed 	: 🗹 »
w_0_start	: 0 0 0 rad/s »
 z_0_fixed 	: 🗆 »
z_0_start	: 0 0 0 rad/s ² »

25. Simulate again for 10 seconds and compare the animations with the previous simulation. What is the difference?

Boundary conditions and Initialization: Thermodynamic system

Mass flow calculation

- 1. In W2, create a model called ValveOpening.
- 2. Drag in a *Modelica.Fluid.System*. Note that this component will automatically be called **system** and get the prefix *inner*.
- 3. Drag in two *Modelica.Fluid.Sources.Boundary_pT*, call them **inlet** and **outlet**, respectively. Parametrize the **inlet** pressure to be 2 bar and **outlet** pressure to be 1 bar.
- 4. Drag in a *Modelica.Fluid.Valves.ValveCompressible*, call it valve. Set its nominal pressure drop and inlet pressure to 1 bar and set nominal mass flow to 1 kg/s see figure below for reference.

Valve Valve for compres Modelica.Fluid.Val	sible fluids, accounts for choked ves.ValveCompressible	l flow conditi			
INFORMATION		>			
PROPERTIES		Ť			
General Advanced A	ssumptions Variables				
	Parameters				
xtCharacteristic		~			
Fxt_full	: 0.5				
valveCharacteristic		~			
Medium*		~			
Nominal operating point					
p_nominal	: 100000	Pa			
dp_nominal	: 100000	Pa			
m_flow_nominal	: 1	kg/s			

5. Declare a real variable *opening* and set it to 1 (*Real opening = 1 "Valve opening";*) and drag in a *Modelica.Blocks.Sources.RealExpression*, that outputs the variable *opening*. Connect it to the *valve opening*. Also connect *port_a* of the **valve** to the **inlet**. See figure below.



6. Drag in a pipe, *Modelica.Fluid.Pipes.DynamicPipe*, and parametrize it with a length of 10 m, diameter of 0.1 m, a height difference between the ports of 10 m. Under the *Advanced* tab of **pipe**, set the model structure to be *port_a - volume - flow model - port_b*, and under the *Initialization* tab set the start temperature to 293.15K. Connect *port_a* of the **pipe** to the **valve** and *port_b* to the **outlet**. The system should now look like the figure below.



7. Now let us set a medium for all the components. Set the medium for **inlet** to **Water using IF97 standard, explicit in** *p* **and** *h*, and click on the propagate button. See the figure below.

Boundary with p Modelica.Fluid.Si	rescribed pressure, temperature, ources.Boundary_pT	composit
INFORMATION		>
PROPERTIES		Ť
General Variables		
use_p_in	: 🗆 »	
use_T_in	»	
use_X_in	. 🗆 »	
use_C_in	. N	
р	200000	Pa
Т	Medium.T_default	K
Medium*		^
WaterIF97_ph		⊸ 🄈

8. Simulate the system using default settings. What is the mass flow through the valve?

Boundary condition 1

- 9. Duplicate the ValveOpening model and call it ValveMassFlow.
- 10. In the declaration of the variable opening, remove its value (*Real opening "Valve opening";*)
- 11. Add an equation *valve.m_flow = 0.1;*Note that at this point, we have the same number of equations in the system.

```
1 model ValveMassFlow
      inner .Modelica.Fluid.System system annotation(....);
      .Modelica.Fluid.Sources.Boundary_pT inlet(p = 200000,nPorts = 1,redeclare replaceable package Medium =
3
      .Modelica.Media.Water.WaterIF97_ph) annotation(••••);
      .Modelica.Fluid.Sources.Boundary pT boundary(nPorts = 1, redeclare replaceable package Medium =
4
      .Modelica.Media.Water.WaterIF97 ph) annotation(•••);
5
      .Modelica.Fluid.Valves.ValveCompressible valve(p_nominal = 100000,dp_nominal = 100000,m_flow_nominal =
      1, redeclare replaceable package Medium = .Modelica.Media.Water.WaterIF97_ph) annotation(....);
6
      Real opening "Valve opening";
      .Modelica.Blocks.Sources.RealExpression realExpression(y = opening) annotation(....);
8
      .Modelica.Fluid.Pipes.DynamicPipe pipe(length = 10, diameter = 0.1, height ab = 10, modelStructure =
      .Modelica.Fluid.Types.ModelStructure.av_b,T_start = 293.15,redeclare replaceable package Medium =
      .Modelica.Media.Water.WaterIF97 ph) annotation(....);
9 equation
      valve.m flow = 0.1;
10
      connect(pipe.port_b,boundary.ports[1]) annotation(....);
                                                                 connect(realExpression.v.valve.opening)
11
      annotation(Line(points = {{-19,24},{-10,24},{-10,8}},color = {0,0,127}));
      connect(valve.port_a,inlet.ports[1]) annotation(....);
12
13
      connect(valve.port_b,pipe.port_a) annotation(....);
14
15
      annotation(....):
16 end ValveMassFlow;
```

12. Simulate the system. Compare the **valve** opening and mass flow through the **valve** to the previous exercise. Which variables are known and unknown in the equation system?

Boundary condition 2

- 13. Duplicate the ValveMassFlow model to a model called ValveMassFlowAir.
- 14. Change the medium model in all components to "Air as mixture of N2 and O2." You can change the medium in a single component, say inlet, and use the *propagate* button.
- 15. Simulate the model. Compare the valve opening to the previous exercise. Is it different? Have we changed any equations in the components?

Boundary condition 3

- 16. Extend the model ValveOpening to a model called ValveOpeningMoon.
- 17. Set the gravity in the **system** component so that the gravity is as on the moon, 1.622 m/s^2 , see figure below.



18. Simulate the system and compare the mass flow to the mass flow in the model **ValveOpening**. Are they different, and if so, why are they different? Have we changed any equations in the components? Have the boundaries of the system changed?

Initialization

19. Extend the model ValveOpening to a model called ValveOpeningSteadyStateInit

20. Under the *Assumptions* tab of the system component, change the *energyDynamics* to *Modelica.Fluid.Types.Dynamics.SteadyStateInitial*, see the figure below.

System System properties and default values (ambient, flow direction, Modelica.Fluid.System									
INFORMATION	>								
PROPERTIES	~								
	Ŧ								
General Assumptions Init	alization Advanced								
Pa	rameters								
allowFlowReversal	✓ >>								
D	ynamics								
energyDynamics	SteadyStateInitial Dyna 🗙 💌								
massDynamics 🛛 😭 🕄	energyDynamics 👻								
momentumDynamics	SteadyState Steady state ba 👻								

21. Simulate the system. Compare the mass flow and the pressures in the pipe discretization volumes to the result of the **ValveOpening** model. What is the difference between the two simulations?

Identifying Degrees of Freedom

- 1. Study the systems (a-d) below. How many degrees of freedom and how many states does each system have? Is there a unique set of states?
 - a. Two inertias with a spring-damper element, see figure below.



b. Two inertias with a gear in between, see figure below.



c. Three inertias with a planetary gear, see figure below.



d. Two fixed elements with a spring-damper in between, see figure below.



2. Implement the systems above as new models **DegreesOfFreedom1-4**, simulate, and inspect the compilation diagnostics, what states were selected? You can see what states have been selected under **State variables** in the advanced diagnostics.

SIMULATIONS			Model Statistics for ImpactTrainingSolutions.Day2				
	∎∎ Result 1			16:55 Show simulation log Show compilation log	Note: Some diagnostics are on <u>4 warnings 0 errors</u>	ote: Some diagnostics are omitted since the compiled model contains er warnings 0 errors	
			Ŧ	Download result	 Model Structure 	1	
	🗌 Only st	€	View diagnostics	 State variables 		4	
		1	Rename	 Initialization equation blocks Equation blocks 		0 linear / 0 non-linear	
CALCULATED VALUES			I		Delete		0 linear / 0 non-linear

Do not forget to activate "generate_html_diagnostics" in execution settings to get access to the advanced diagnostics.

Inverting system models

Create a new model called InverseSystem and build the system below. Use J=0.1 kg.m² in inertia1; c=10 N.m/rad, d=0.1 N.m.s/rad in springDamper; J=1 kg.m² in inertia2; and set *freqHz*=2 Hz, *damping*=0.5 s⁻¹ for expSine. The square block in the bottom is a *Modelica.Block.Math.InverseBlockConstraints*. The model should be like the figure below.



- 2. Simulate for 10 seconds and plot the torque (torque.tau) and angle (angleSensor.phi).
- 3. What problem does the model above solve?

This concludes workshop 2.2. Well done!

