Exploring Model Structure with Equation Incidence

Before a Modelica model is simulated, Dymola translates it into a causal model. The non-causal equations are sorted and an execution order is determined. Numerical methods, like Dassl, can then more easily integrate the resulting, translated model.

In Dymola there are several ways to obtain information about a translated model. For example, the translation log contains summaries about variable types, selected states, systems of equations, and initial values, etc.

To investigate relationships between variables in the generated execution order, *Plot Dependencies* is often used. In Dymola 2021x we introduce a graphical complement: the *Equation Incidence* view.

Introduction

Learning about the structure of a translated model often gives the modeler useful insights, especially when a simulation is slow or fails. For slow simulations, the *Simulation Analysis* tool may indicate specific problematic variables. Similarly, when a simulation fails, the simulation log may point to certain variables causing the problem. As a modeler, the question may then arise:

– I know which variables cause problems, but what should I do with this information?

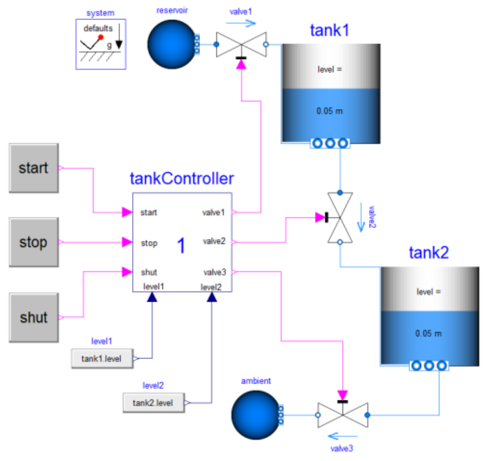
An answer is to learn more about the translated model and how the problematic variables are computed. Combining this insight with knowledge about the original Modelica model and the modelled physical system often helps in reaching a solution.

As scanning through the full translated model may be a daunting task for large models, Dymola offers several ways to extract the most relevant information and to present it in a user-friendly manner.

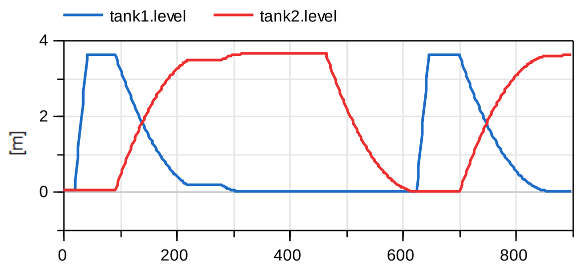
The *Equation Incidence* view is an interactive and graphical tool for exploring the structure of a translated model.

Controlled Tanks example model

To introduce the *Equation Incidence* view we will use the example model *ControlledTanks1* from the *Modelica_StateGraph2* library – a free library for modelling discrete-event, reactive, and hybrid systems.

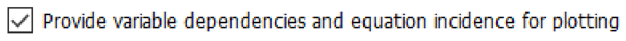


The example model consists of two tanks, three valves, and a state machine controlling the valves. During simulation, `valve1` opens to fill up `tank1` with water from the `reservoir`. When `tank1` is full its content is drained into `tank2`, making a pause when the former tank is almost empty. Finally, `tank2` is emptied and the cycle repeats.

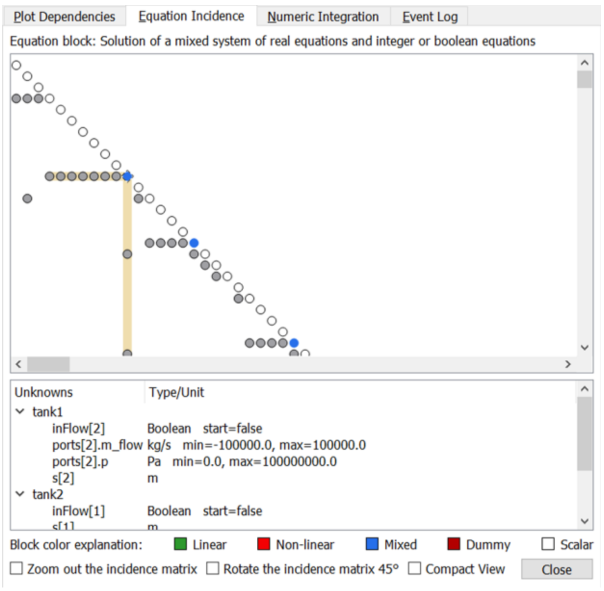


Exploring the equation incidence

To get started navigate to the *Debug* tab of the *Simulation Setup*. The following check box tells Dymola to generate additional structural information during translation. This enables both *Plot Dependencies* and *Equation Incidence*.


☑ Provide variable dependencies and equation incidence for plotting

After a model has been simulated the *Equation Incidence* view can be found in the *Simulation Analysis* widget (located under the *Simulation* ribbon).



Equation block: Solution of a mixed system of real equations and integer or boolean equations

```
Unknowns              Type/Unit
v tank1
     inFlow[2]        Boolean   start=false
     ports[2].m_flow  kg/s   min=-100000.0, max=100000.0
     ports[2].p       Pa   min=0.0, max=100000000.0
     s[2]             m
v tank2
     inFlow[1]        Boolean   start=false
     s[1]             m
```

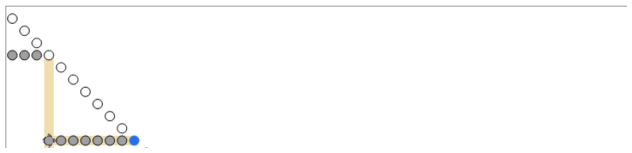
Block color explanation: 🟩 Linear 🟥 Non-linear 🟦 Mixed 🟥 Dummy ☐ Scalar

☐ Zoom out the incidence matrix ☐ Rotate the incidence matrix 45° ☐ Compact View Close

The window shows the dependency graph for *ControlledTanks1*. The graph's diagonal contains either scalars, which are represented by white circles, or systems of equations, marked by colored circles. In the current example there are three blue circles indicating that there are three systems with mixed continuous and discrete equations. By selecting the first system, we get a list of all variables it solves for. This includes iteration variables, torn variables, as well as discrete variables. The list tells us that the first system solves for the flow and pressures around `valve2`, between the two tanks.

The grey nodes to the left of the blue node represent variables that need to be computed before the system can be solved. The grey nodes below it show where variables from the system are used in subsequent computations. The node `valve2.open`, selected below, exemplifies a downstream dependency.

Equation block: valve2.open = (tankController.fillTank2.active or tankController.emptyTanks.active or tankController.shutTank1.active) and tank2.level <= 1.01*tankController.limit2;
-> Solution of a mixed system of real equations and integer or boolean equations



To illustrate the power of the graphical presentation, we ask the question: *How do the systems of equations depend on each other?* Following the grey nodes in the first figure above, we can see that the systems are independent. For example, no solution of the first system is needed to solve the second system. In this case, it is easy to see, as there are no grey nodes above the second system in the vertical yellow line.

Debugging a simulation failure

The outlet port of `tank1` is placed 1 cm above the floor of the tank. Let's move it to the very bottom of the tank: `tank1.portsData[2].height = 0.0;`

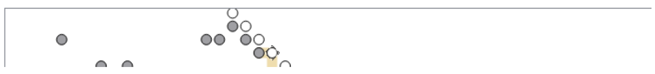
The result is a simulation failure just as `tank2` becomes empty. The *Event Log* (and the previously obtained model knowledge) tells us that `valve1` had just opened to start filling the empty `tank1`.

The simulation log explains that there were large numeric errors when restarting after the event. Further, it points us to the derivative of the water temperature in the first tank, `der(tank1.medium.T)`.

```
Final values of states and state derivatives:
State              ,      State value, Derivative value
tank1.level        ,     4.03685e-16,         0.160621
tank1.medium.T,              293.267,      2.08587e+12
tank2.level        ,             0.01,                0
tank2.medium.T,              293.275,                0
```

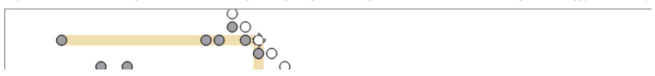
The *Equation Incidence* view tells us that Dymola computes this state derivative as a constant multiplied by the derivative of the specific internal energy of the water, `der(tank1.medium.u)`.

Equation block: der(tank1.medium.T) = 0.0002390057361376673*der(tank1.medium.u);



Following the downstream dependency, we see that computing `der(tank1.medium.u)` is more complex.

Equation block: der(tank1.medium.u) = -(der(tank1.m)*tank1.medium.u-der(tank1.U))/tank1.m;



Indeed, this computation is derived from the equation for the *total* internal energy of the water in `tank1`. (Note that Dymola had to differentiate this equation to reduce the index before generating the code for computing `der(tank1.medium.u)`.)

Here, we also see the source of the simulation failure. There is a division by the water mass in `tank1`. Double-clicking `der(tank1.medium.u)` opens it in *Plot Dependencies*, from there we can easily plot the mass `tank1.m`. It is zero when the simulation fails!

The default parameterization avoids the division by zero as there is always, at least, one centimeter of water remaining in `tank1`.

Conclusion

The *Equation Incidence* view offers a new, powerful tool to understand the model structure and helps debugging failing or slow simulations.