



How to Automate Modelica Models Experimentation in Dymola

Raees B. K. Parambu -- Systems (Project) Engineer

Dymola Version Used: Dymola 2018

Keywords: Automate, Dymola, Modelica, Model, Experimentation, Parametric sweep.

Abstract

This article aims to describe a methodology for automating the experimentation of Modelica models in Dymola. For this purpose, a simple RC circuit model is adopted from a previous blog article. With the aid of a specialised Modelica function class and built-in Dymola functions, the chosen model is modified to perform an automated parametric sweep study and to plot the simulation results.

Introduction

Dymola (Dynamic Modelling Laboratory) [1] is an advanced systems-engineering tool. It is not only useful for modelling multi-engineering & complex physical systems but also includes powerful postprocessing (experimentation, plotting and animation [2]) features for analysing the simulation results in a convenient and user-friendly manner. Dymola is based on the open source Modelica (equation-based object-oriented) modelling language. In addition to the Modelica Function (a specialised Modelica class) available in the Modelica Language (see specification guide [3]), Dymola also provides several built-in functions. These built-in functions are classified into the following function groups: Help, Simulator API, System, Plot, Trajectories, Animation, Matrix IO, Documentation and others [4]. This article utilises some of these built-in functions together with a Modelica Function to describe a methodology to automate the experimentation of Modelica models in Dymola.

A parametric sweep study usually involves a model to simulate several times with a selected set of parameters by which to analyse the sensitivity of the model in relation to its parameter values. A simple parametric sweep study is given in the previous blog post of '[How to simulate a model multiple times with different parameter values](#)' on the Claytex [TECH BLOG](#) [5]. This blog post [5] showed two distinct methods; the first one is using the '`simulateExtendedModel`' and the second one is using the '`translateModel`' built-in functions available in 'DymolaCommands' package. On the one hand, the blog article [5] used the Dymola Script editor to run a simple test model multiple times and to store the simulation results of each iteration separately, but on the other hand, the article [5] did not show how to plot the simulated results. Therefore, the present article will not only fill this gap but also use the Modelica function instead of the scripting facilities of Dymola for demonstrating a model experimentation method in Dymola.

Methodology

The methodology provided in this article is given in two parts:

- (1) Briefly, describes a simple RC circuit model which is adopted from a previous blog post and its modification.
- (2) Shows a methodology to create a Modelica function to automate the model experimentation.

Model Selection

The Modelica modelling language uses an acausal (equation) based object-oriented (OO) modelling technique rather than a causal block-based modelling method [6]. One of the advantages of an acausal modelling technique is to be able to utilise the concept of inheritance and abstraction in the OO modelling language [3]. A simple example of such an acausal model is presented in [7]. The schematic of the RC circuit model from [7] is copied below.

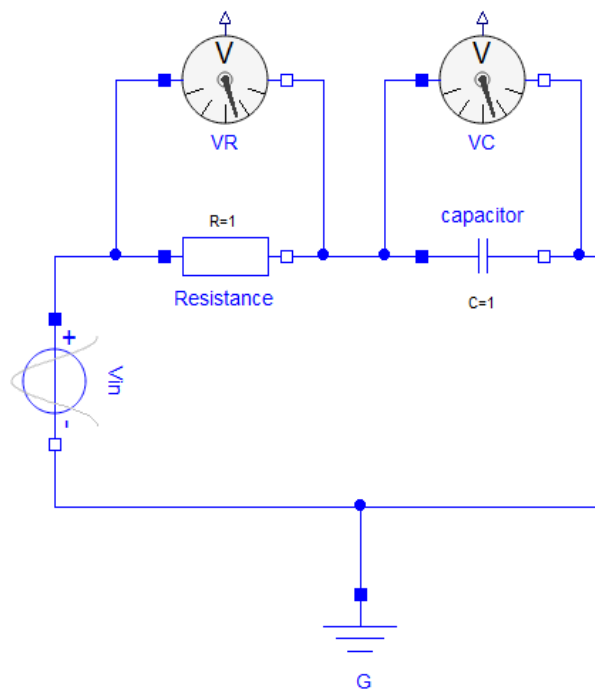


Figure 1- The RC network is depicted as an acausal equation-based model in Dymola [7].

In [7], a state-space model of a simple RC circuit is presented, and a methodology is shown to develop this model from scratch. Using this model, an experimental study was also conducted in Dymola which investigated the use of this simple RC circuit model for frequency filtering applications. However, in [7] both the model experiments and the plotting of the simulated results were done manually. In the present article, the author adopts and modifies this model to integrate it with a Modelica function shown in the next section. A screenshot of this modified Modelica model code is provided in the appendix. Compared to [7] this code has two **Added Sections**; one below the `extends` command and other below the `equation` section.

Create a Modelica function

This section describes how to build a Modelica function called “ModelExperiment”. ModelExperiment is developed to automate the experimentation of the model mentioned above, within Dymola. A set of procedures are coded in the created ModelExperiment function and has four main stages. Each of these stages also has sub-tasks.

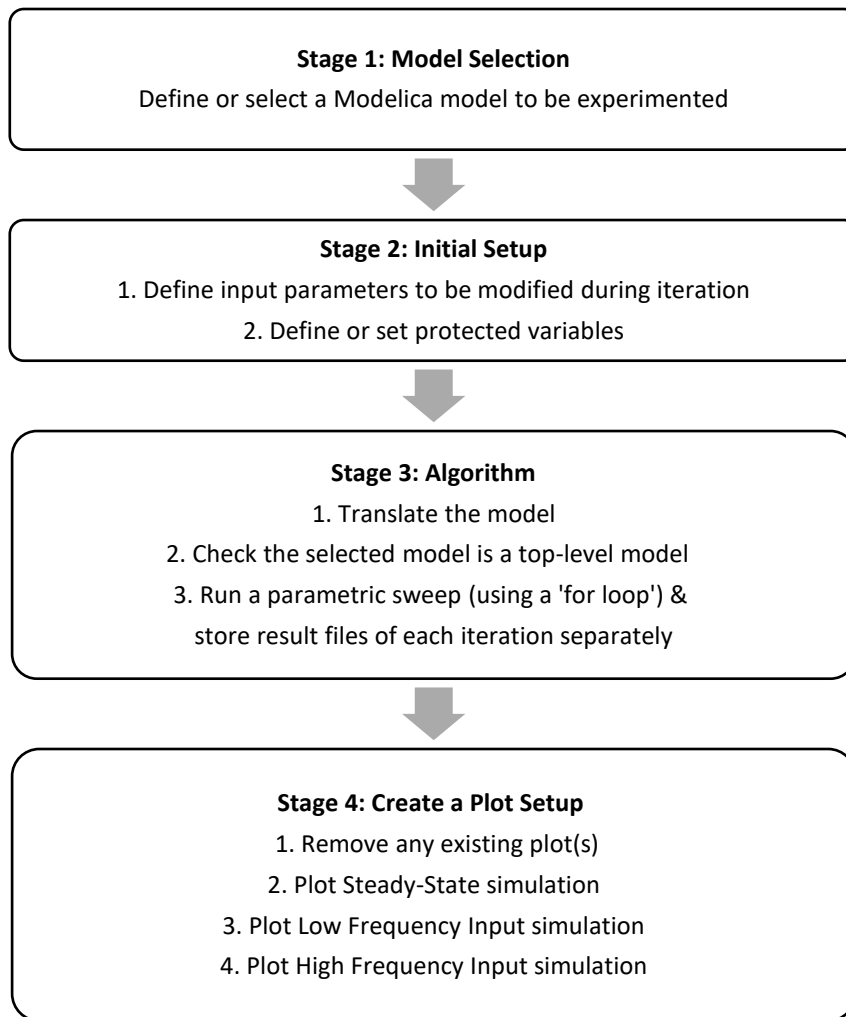


Figure 2- Flow chart of the created ModelExperiment function.

Figure 2 depicts the sequential stages of the ModelExperiment function. By right clicking on the ModelExperiment in Dymola Package Browser, and then selecting ‘Call Function’, this will generate the following dialog box:

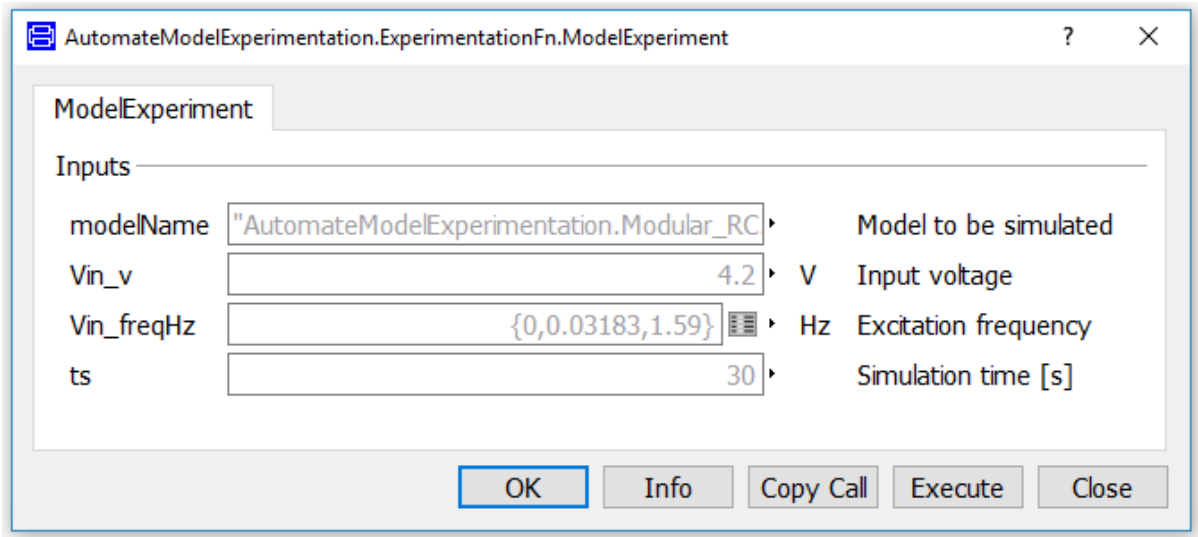


Figure 3- Dialog box of the ModalExperiment function.

In Figure 2, the automation process begins with Stage 1: defining or selecting an existing Modelica model within Dymola. In our case, this model is the modified RC circuit mentioned in the previous section. This step is also shown in the first input field **modelName** of the ModelExperiment function dialog in Figure 3.

Stage 2 in Figure 2 describes the Initial Setup. It has two sub-tasks which are as follows;

- 1) The user can define or change the input parameter values. Figure 3 also represents this Initial Setup stage, which includes three input fields that can be modified later: Vin_v, Vin_freqHz and ts.
- 2) Define or set the protected variable(s): in Modelica, this variable has multiple purposes. For more details about this type of variable, please refer to [3]. In the context of this code, only one protected variable is used for performing a Boolean operation in sub-task 2, Stage 3 below.

Please note that sub-task 2 of Stage 2 and the rest of the stages in Figure 2 can only be accessed within the ModelExperiment function.

There are three sub-tasks within Figure 2, Stage 3:

- 1) Translate the selected model using Dymola built-in function: `translateModel`.
- 2) If the model is translated successfully, say a top-level model [8], then using the Modelica built-in function, `Modelica.Utilities.Streams.print`, we can print the status to Dymola Command log window in the Dymola Simulation tab.
- 3) Run a parametric sweep of the selected model using three different values of Vin_freqHz and store the corresponding simulation results of each iteration in a separate result file.

Figure 2 Stage 4 is dedicated to the Plot Setup. This stage also has four sub-tasks. These sub-tasks are as follows,

- 1) Remove any Diagram or figure opened in the New Plot Window of the Simulation mode of Dymola. This step uses the `removePlots` Dymola built-in function.

- 2) Open a New Plot Window and plot the steady-state simulation result stored in a separate file in Sub-task 3, Stage 3 of Figure 2.
- 3) Open a second plot window and plot the low-frequency input simulation result which was stored in a separate file in Sub-task 3, Stage 3 of Figure 2.
- 4) Again, open a third plot window and plot the high-frequency input simulation result which was also stored in a separate file in Sub-task 3, Stage 3 of Figure 2.

Please note that Sub-tasks 2 to 4 of Stage 4 in Figure 2 are done by using a Dymola built-in function, `createPlot`.

Results & Discussions

For the sake of demonstration purposes, the operations coded in the ModelExperiment function are made very simple. It means that, in ModelExperiment, a user can only select a **Simulation time** for all the experiments at a time. In this article, there are three experiments: Steady-State, Low and High-Frequency Input simulations are considered with an equal Simulation time of 30 s. However, in [7], the High-Frequency Simulation Input study was carried out for 10 s only.

The plot windows generated after the execution of the ModelExperiment function are shown in Figures 4, 5 and 6. For illustrating the discrepancy mentioned in the last paragraph, Figure 5 from [7] is also shown in Figure 7.

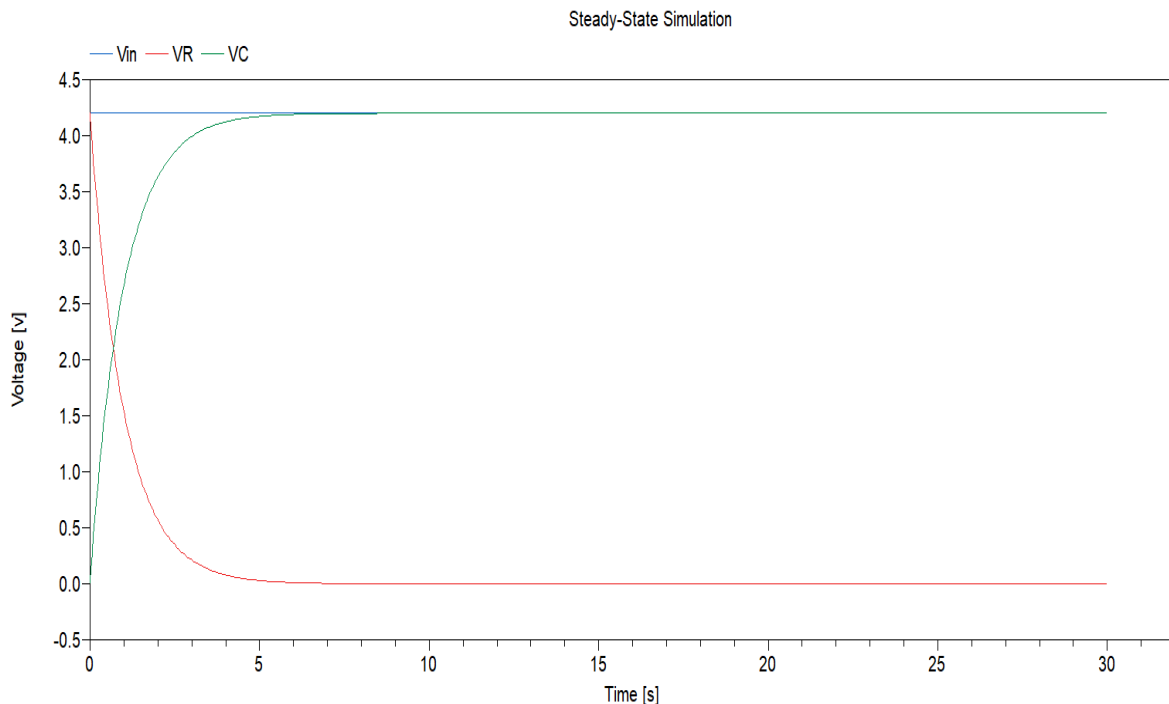


Figure 4- Steady-State simulation plot of the modified RC model.

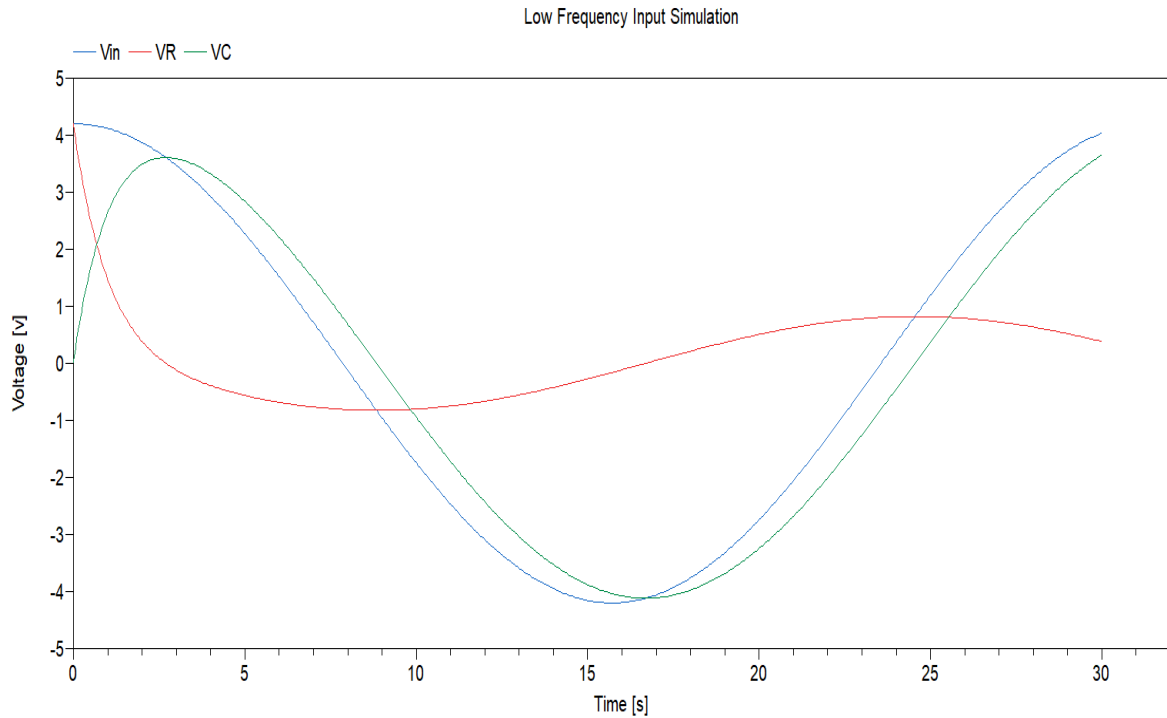


Figure 5- The voltage response of RC circuit at a very low excitation frequency of 0.2 rad/s or 0.03183 Hz.

Both Figure 4 and 5 match with Figure 3 and 4 in [7] for a simulation time of 30 s. Please note that physics of this model and the plots shown from Figure 4 to 6 are beyond the scope of this article. Those who are interested in getting more insight into the physics of the model can refer to [7].

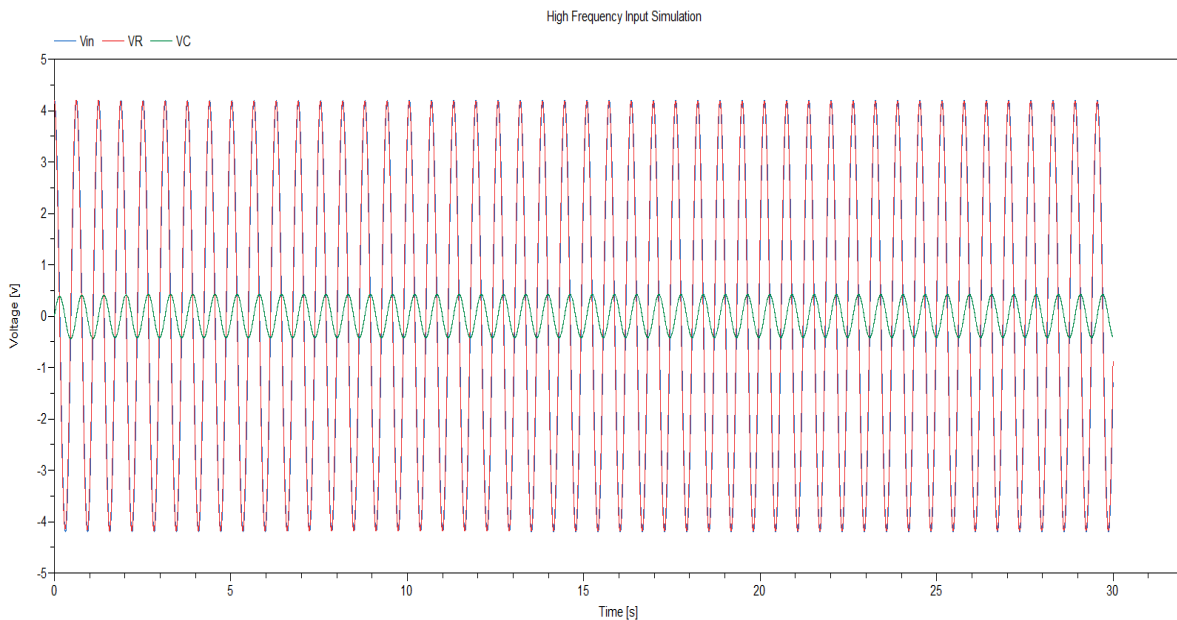


Figure 6- The voltage response of RC circuit at a high excitation frequency of 10 rad/s or 1.59 Hz simulated for 30 s.

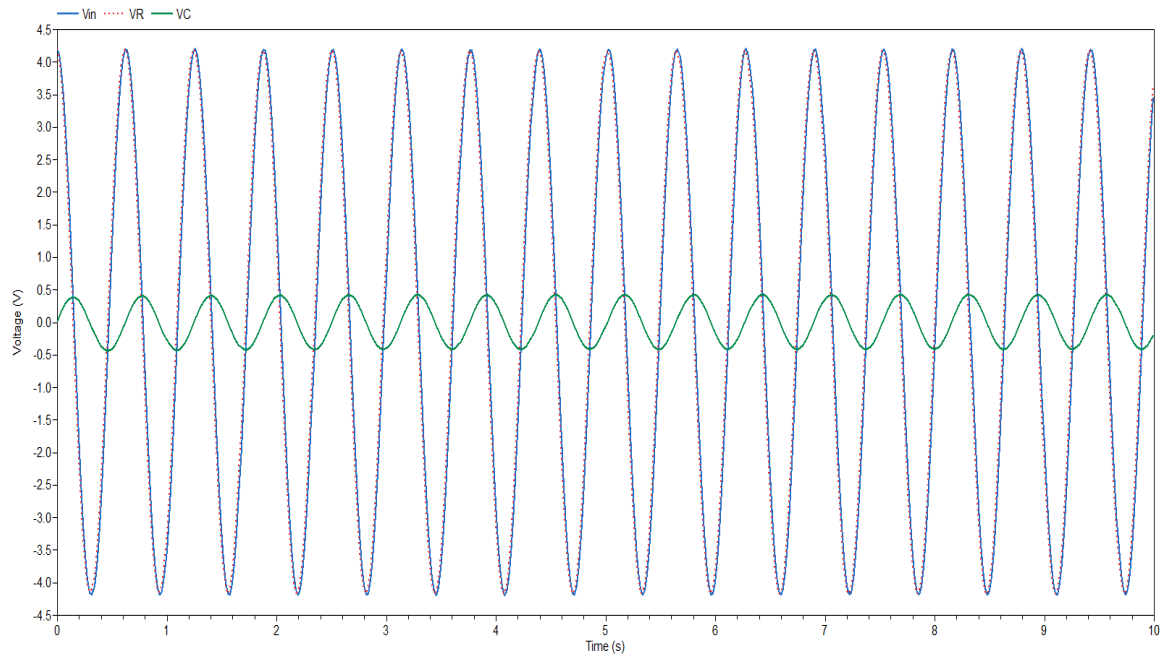


Figure 7- The High-frequency input simulation copied from [7].

Moreover, Figure 6 and 7 represent the same high-frequency experiment, but, simulated with two different simulation times.

In summary, the presented method in this article also proves that Dymola has at least two choices, either using Dymola Scripting or Modelica Function, to perform an automated Modelica model experimentation in a user-friendly manner.

Conclusion

This short article presented a methodology to automate Modelica model experimentation in Dymola. A simple and modular model adopted from the previous blog article helped to demonstrate the automation process. The presented method shows that a user can not only perform an automated parametric sweep study but is also able to plot the simulation results of interest in a more user-friendly way. For more information about how to create a Modelica function to carry out the proposed automation process, please [contact](#) Claytex Services Limited.

References

1. Dassault Systemes, 2018. *Dymola—Dynamic Modeling Laboratory*.
2. Page no. 14, *Dymola Dynamic modelling laboratory, User Manual Volume 1*, version 2018, Copyright © Dassault Systèmes, 1992–2017, March 2018.
3. *Modelica language specifications version 3.3 Revision 1*, Copyright © 1998–2014 Modelica Association July 11, 2014, <https://www.modelica.org/documents> (03 Oct. 17, last accessed).
4. Page no. 701, *Dymola Dynamic modeling laboratory, User Manual Volume 1*, version 2018, Copyright © Dassault Systèmes, 1992–2017, March 2018.
5. *How to simulate a model multiple times with different parameter values*, <http://www.claytex.com/blog/how-to-simulate-a-model-multiple-times-with-different-parameter-values/> (03 Oct. 17, last accessed).
6. Dempsey, M., *Getting more from simulation – Part 3 Component orientated modelling versus block diagram modelling*, <http://www.claytex.com/blog/getting-more-from-simulation-part-3-component-orientated-modelling-versus-block-diagram-modelling/> (03 Oct. 17, last accessed).
7. Raees B. K. Parambu & S. Harrison, *A State-Space Model of a simple RC Network in Dymola & potential Applications*, <http://www.claytex.com/blog/a-state-space-model-of-a-simple-rc-network-in-dymola-potential-applications/> (03 Oct. 17, last accessed).
8. Raees B. K. Parambu, *Handling of Modelica Model's Parameter Values in Dymola*, <http://www.claytex.com/blog/handling-of-modelica-models-parameter-values-in-dymola/> (03 Oct. 17, last accessed).

Appendix – A Modified Modelica Model Code

```
model ModularRC_Modified "Model of modular RC circuit"
  extends Modelica.Icons.Example;
  //Added Section
  output Modelica.SIunits.Voltage V_in "Input voltage";
  output Modelica.SIunits.Voltage V_R "Resistor voltage";
  output Modelica.SIunits.Voltage V_C "Capacitor voltage";

  Modelica.Electrical.Analog.Basic.Resistor Resistance(
    alpha=1e-3,
    i(start=0),
    useHeatPort=false,
    R=1,
    T_ref=293.15) a;
  Modelica.Electrical.Analog.Basic.Ground G a;
  Modelica.Electrical.Analog.Sources.CosineVoltage Vin(freqHz=0, V=4.2) a;

  Modelica.Electrical.Analog.Basic.Capacitor capacitor(C=1)
    a;
  Modelica.Electrical.Analog.Sensors.VoltageSensor VR a;
  Modelica.Electrical.Analog.Sensors.VoltageSensor VC a;
equation

  //Added Section
  V_in = Vin.v;
  V_R = VR.v;
  V_C = VC.v;

  connect(Vin.n, G.p) a;

  connect(Vin.p, Resistance.p) a;

  connect(Resistance.n, capacitor.p)
    a;
  connect(capacitor.n, G.p) a;
  connect(VC.p, capacitor.p)
    a;
  connect(VC.n, G.p) a;
  connect(VR.p, Resistance.p) a;
  connect(VR.n, capacitor.p) a;
  a
end ModularRC_Modified;
```