Initialization of Modelica models in Scicos

Masoud Najafi Ramine Nikoukhah
INRIA-Rocquencourt, Domaine de Voluceau,
78153, Le Chesnay Cedex, France

Abstract

The current Scicos Modelica compiler only supports one form of initialization of index-1 DAEs: differential state variables and system parameters are considered known, algebraic variables as well as the derivatives of differential variables are computed. In many practical applications this is not enough. For example, often it is required to start a simulation in an equilibrium state. Scicos has recently been extended to support general initialization of Modelica models. In this paper, we present this extension and in particular the way the continuous-time part of a Modelica program is initialized, the graphical interface, as well as the initialization methods used.

Keywords: hybrid differential equations; initialization; numerical solver; Modelica; Scicos

1 Introduction

The Modelica compiler used in Scicos[1] [1, 2] has been developed in the SIMPA[2] project with the participation of INRIA, IMAGINE, EDF, IFP, and Cril Technology. Recently the ANR[3]/RNTL SIMPA2 project has been launched to develop a more complete Modelica compiler. The main objectives of this project are to extend the SIMPA compiler to fully support inheritance and hybrid systems, to give the possibility to solve inverse problems by model inversion for static and dynamic systems, and to enhance initialization of Modelica models.

A model can be simulated only if it is initialized correctly. The reason lies in the fact that a DAE (Differential-Algebraic Equation) resulting from a Modelica program can be simulated only if the initial value of all variables as well as their derivatives are known and consistent. An index-1 DAE can be formulated as

$$0 = F(x', x, y, p)$$

where x, x', y, p are the vector of differential variables, derivative of differential variables, algebraic variables, and model parameters, respectively [3, 4].

Previous version of the Modelica compiler of Scicos supports only one form of initialization; it assumes that the parameters and the initial value of differential variables (variables whose derivative exist in the Modelica program) are known and given by `start` keyword, then a solver is used to compute the derivative of differential variables and algebraic variables. The user can also give start values of algebraic variables in the Modelica program which are used as guess values to help the solver to find consistent initial values. In many practical applications, this approach of initialization does not cover all forms of initializations. For example, often it is required to start a simulation in the steady state. In this case, derivatives of differential variables are set to zero and the initial values of all the variables are found as a function of known outputs (or observable states). The initialization problem can also be formulated as an inverse problem where the system outputs are known at initial time, and the inputs and internal states are to be computed. Sizing is another form of initialisation, where, *e.g.,* a parameter of the model is computed at steady states as a function of a system output (or observable states).

The next version of the Modelica compiler used in the Scicos simulator provides support for a more general form of initialization. In this initialization methodology, the user can select freely the knowns and unknowns of the initialization problem between the all Modelica variables, the derivatives of differential variables, and the parameters of the model. A confidence factor is associated to each value specifying the degree of confidence in that value. The confidence factor goes from zero to one where zero means the value is just a pure guess and one means the guess value corresponds to the actual value. This latter situation corresponds in particular to the usage of the "fixed" keyword in Mod-

[1] www.scicos.org
[2] Simulation pour le Procédé et l'Automatique
[3] French National Research Agency

elica. With this initialization methodology, no matter if an unknown is a parameter or a differential or algebraic variable, it can be computed as a function of other known values.

Another problem with the previous approach of initialization concerns with start values of algebraic variables. The Modelica compiler of Scicos performs formal simplifications on Modelica programs and generates a DAE which is used for both model initialization and dynamic simulation. Using the simplified Modelica model causes a problem in initialization of big Modelica programs where guess values of algebraic variables are important for convergence of the solver. The generated DAE contains all differential variables and their derivatives, but many algebraic variables are eliminated during the formal simplification phase of the Modelica model. This has several consequences: First, algebraic variables to be eliminated may change when the Modelica model changes a bit. Thus, the user is unable to know in advance what algebraic variables will be selected to provide their guess values. Furthermore, in many cases the remained algebraic variables may not have physical interpretation and the user cannot provide the guess value without performing a calculation.

Because of these difficulties, we decided to separate the model initialization and the dynamic simulation and generate two DAEs for each one:

- a flat Modelica model without formal simplification to be used in the initialization phase
- a simplified Modelica model to be used in the dynamic simulation phase.

With this approach, the problem of the selection of algebraic variables no longer exists. In fact, once the initialization phase finished, initial values of all algebraic variables as well those of derivatives of differential variables are known. As a consequence, the selection of algebraic variables in the simplified Modelica code does not affect the dynamic simulation.

In Modelica, the `start` keyword can be used to set start values of variables. Start value of derivatives of variables can be given within `initial equation` section. For small programs, this method can be easily used but as the program size grows, it becomes difficult to set start values and change the `fixed` attribute of variables/parameters directly in the Modelica program; initialization via modifying the Modelica model is specially difficult for models with multiple level of inheritance. *e.g.,* The GUI (Graphical User Interface) of Dymola[4] is not well adapted to visualize and fix/relax the variables and the parameters. Then, the user often needs to have a single model but with several initialization scenarios. So, for each scenario a copy of the model should be saved. Furthermore, it is not possible to use variable attributes used in a model in another model.

Having confronted these inconveniences, we found it easier and more intuitive if start values and other attributes of variables/parameters are provided via a GUI. In the GUI, the user can easily change attributes such as `start`, `fixed`, `max`, `min`, `nominal` for variables/parameter of a model. Furthermore, it is possible to indicate whether a variable, the derivative of a variable or a parameter must be fixed or relaxed during initialization. It is also possible to save the resulting configuration in a file and use it later.

In the following sections the initialization methodology for Modelica models, the initialization GUI, and available initialization computing methods for continuous-time Modelica models (index-1 DAE) will be explained.

2 Initialization methodology for Modelica models

The first objective of the initialization is to compute consistent initial values for all variables and derivatives of differential variables (for index-1 DAEs). In order to have access to all variables, the model should be flattened, *i.e.,* a model without inheritance should be constructed. Then, in order to initialize the model, which is often a dynamic model, *i.e.,* it contains derivatives, it should be converted into a pure algebraic model. For that, all derivatives should be replaced by algebraic variables.

The procedure of initialization is given in Figure 1. The initialization is composed of:

- converting the Modelica program into a flat Modelica model
- converting the flat model to an XML file
- modifying the XML file by the initialization GUI
- converting the XML file back to a Modelica program
- computing the unknown variables/parameters.

[4] www.dymola.com

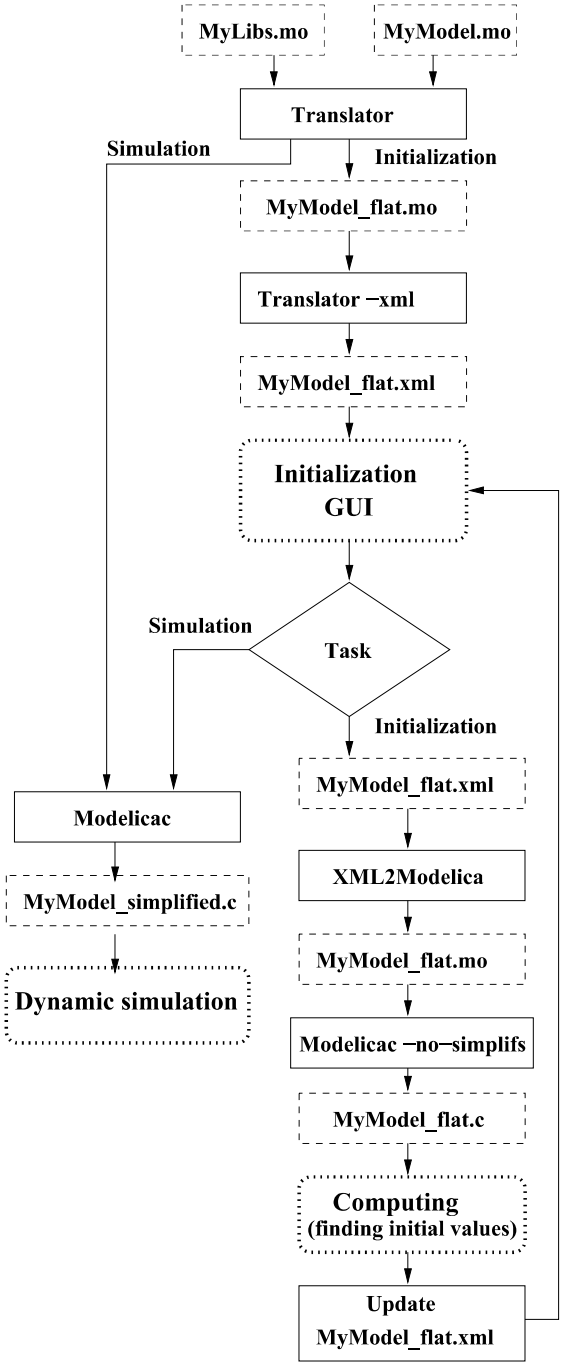

Figure 1: Initialization in Scicos

In the initialization process, three external applications are used: `Translator`, `XML2Modelica`, and `modelicac` (all developed at *LMS Imagine.Lab*[5]). `Translator` is used for three purposes:

- Modelica Front-end for dynamic simulation. When called with appropriate options, `Translator` generates a flat Modelica program. For that, `Translator` verifies the syntax and semantics of the Modelica program, applies inheritance rules, generates equations for `connect` expressions, expands `for` loops, handles predefined functions and operators, performs the implicit type conversion, and etc. The generated flat model contains all the variables, derivatives of differential variables, and parameters defined and declared with `fixed=false`. Constants and parameters with the attribute `fixed=true` are replaced by their numerical values.

- Modelica Front-end for initialization. In this case, besides generating a flat Modelica model, derivatives of the variables are replaced by an algebraic variable. As a convention, *e.g.,* "$der(x)$" is replaced by "$\_\_der\_x$" in the generated flat model for initialization.

- XML generator. When called with -xml option, `Translator` generates an XML file from a flat Modelica model. The generated XML file contains all the information of the flat model. An example of an XML file is given in Appendix A.

Once the XML file generated, the user can change variable/parameter attributes. The modified XML file should be reconverted into a Modelica program to be compiled and initialized. `XML2Modelica` is used to perform this task.
`Modelicac`, which is a compiler for the subset of the Modelica language, compiles a flat Modelica model and generates a C program for Scicos targets. The main features of the compiler are the simplification of the Modelica models and the generation of the C program ready for simulation. It supports discontinuous model switching and provides the analytical Jacobian of the model. It does not support higher index DAEs. This compiler is used in initialization and dynamic simulation stages:

- when called with `-no-simplifs` option, `modelicac` generates a C program without doing formal simplification and variable elimination. Only parameters with the attribute `fixed=true` are eliminated and replaced by their numerical values.

- when called with appropriate options, `modelicac` can perform formal simplification and generate a simplified C program for dynamic simulation.

When the user requests a Modelica initialization in Scicos, as shown in Figure 1, `Translator` is called

[5] http://www.lmsintl.com

and first a flat Modelica model, then an XML file are generated. The XML file can then be used in the initialization GUI. The user can change the variable/parameter attributes in the XML file. The modified XML file is then translated back to a Modelica program. The Modelica program is compiled with `modelicac` and a C program is generated. The C program is used by Scicos to compute the initial value of variables/parameters. Once the initialization finished, whether succeeded or failed, the XML file is updated with the most recent results which the user can visualize and decide if the dynamic simulation should start or not.
At the beginning of the dynamic simulation, the initial values of the variables are read from the XML file and the simulation can start. The results of the dynamic simulation can also be saved in an another XML file to be used as a starting point for another simulation.

3 Initialization GUI

In order to manipulate an XML file, a GUI has been developed for Scicos. This API has been developed in TCL/TK[6]. A screenshot of this GUI is shown in Figure 2.
With this GUI, the user can Open/Merge/Close/Save XML files and visualize the attributes of variables/parameters. Each variable/parameter has several attributes: `name`, `id`, `type`, `fixed`, `value`, `weight`, `max`, `min`, `nominal`, `comment`, and `selection`.

- `name:` attribute is the name of the variable/parameter used in the Modelica program. Note that the derivative of a variable is replaced by an algebraic variable. The user cannot change the `name` attribute.
- `id:` the identification of `name` and is used to locate the variable in the XML file. The user cannot change the `id` attribute.
- `type:` it indicates whether `name` is a parameter or a variable in the original Modelica program. The user cannot change the `type` attribute.
- `fixed:` it shows the value of the `fixed` attribute of `name` in the original Modelica program. The user cannot change the `fixed` attribute.
- `value:` it is the default value of `name` in the original Modelica program. The user can modify this field.
- `weight:` it is the confidence factor attributed to `name` and can take a value in the range $[0,1]$. `weight==0` means the value is just a pure guess. This situation corresponds to the `fixed=false` in Modelica. `weight==1` means the given value corresponds to the initial value. This situation corresponds to `fixed=true` in Modelica. The default value of `weight` for parameters and differential variables is one, whereas for algebraic variables and derivative of differential variables (converted to variables) is zero. Note that when the user sets `weight` to one, the corresponding variable/parameter will be considered as a constant and in the initialization phase, it will be replaced by its numerical value. In this way, a parameter in Modelica program can be considered as an unknown and its value be computed during the initialization phase.

  For large models, it is nearly impossible to give all guess values, so the guess values of unspecified variables are set to zero by the compiler. Furthermore, many variables are redundant and the user does not know the initial guess of which ones should be given. This often happens with variables linked by the `connect` operator. Consider, *e.g.,* the following equation set.

$$F(X): \begin{cases} 0 &= \dfrac{x-3}{(x-3)^2+1} - 0.1 \\ 0 &= x-y \end{cases}$$

  In this case, if the user sets the initial guess of y to 10 and leaves the guess value of x unspecified *i.e.,* $x=0$, although $y=10$ is close to the solution, the Newoton's methods will likely fail. Because, the solver ignores the initial value of y and uses that of x. In fact, there is no way to indicate the solver the guess value which is "more" correct than the others. Using the *weight* attribute can be useful to help the solver to give more importance to the values specified by the user and help the solver to converge toward the desired solution.

  When the final model is not square or *weight* values are are zero or one, an optimization problem should be solved to minimize the cost:

$$\texttt{Cost} = \lambda\ F(X)F^t(X) + \sum_{i=1}^{N} w_i(x_i - v_i)^2 \qquad (1)$$

  where x_i is an unknown, v_i is its guess value, and w_i is *weight* or the confidence factor. λ is the Lagrange multiplier and N is the total number of unknowns.

[6] http://www.tcl.tk/software/tcltk/

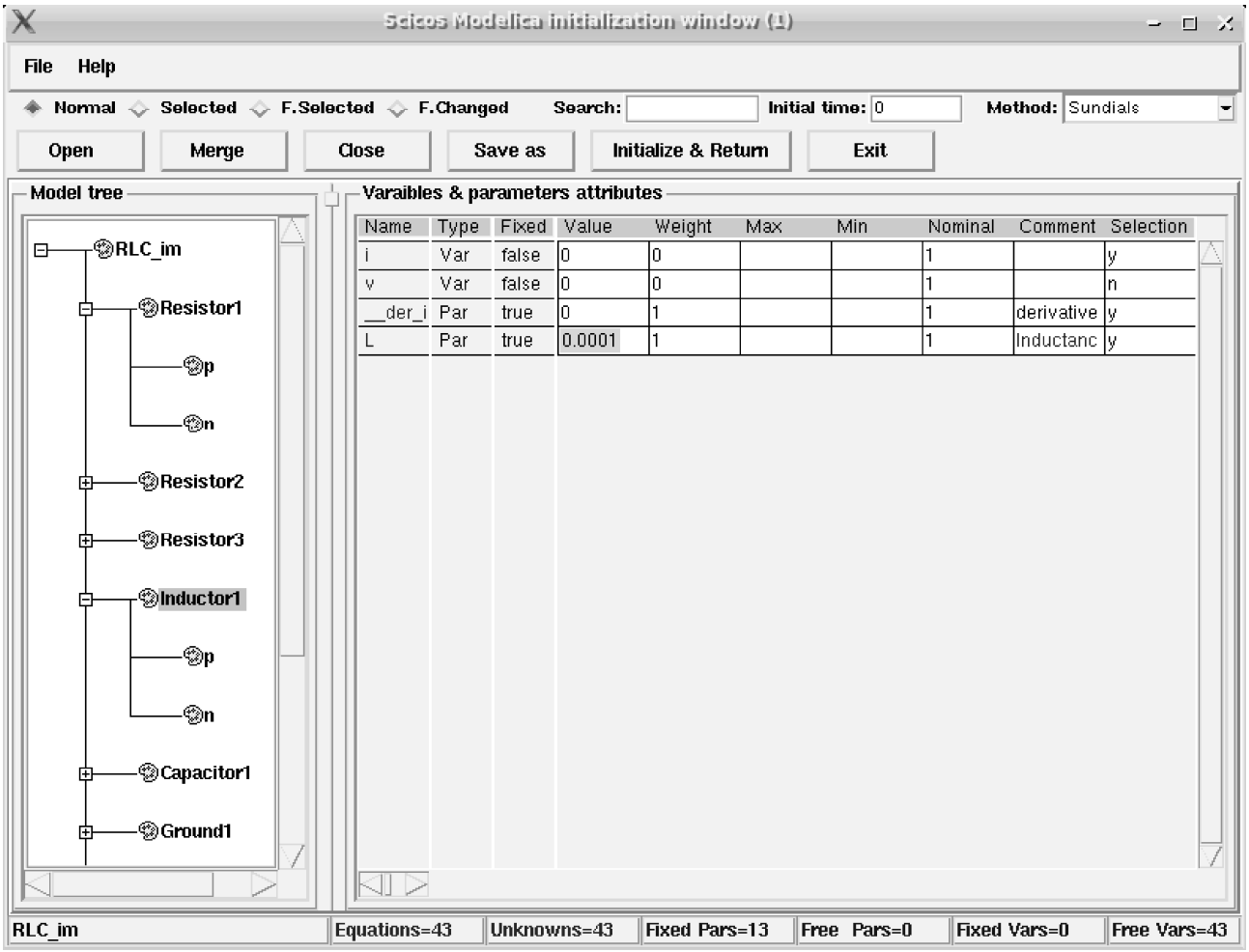

Figure 2: Screenshot of the initialization GUI in Scicos for the electrical circuit of Figure 4

- `max/min:` they are used to set maximum and minimum bounds on values.
- `nominal:` it is used to set error tolerances and normalize variables/parameters.
- `Comment:` it is the comment provided in the Modelica program for variables or parameters and can be modified by the user.
- `selection:` it is used to select and display interesting variables/parameters. If the display mode is `Selected`, only variables/parameters whose selection field is 'y' will be displayed. This option is useful specially when a block has many variables but the user is interested only in a few. There are other display modes, *i.e.,* `F.selected` which displays all selected variables, `Changed` which displays only the variables or parameters whose `weight` attributes have changed.

Once the user changes the new attributes of variables/parameters, the initialization process can be invoked by clicking on the "Initialize& Return" button. The obtained results, either successful or failed, are put back into the XML file and new values are displayed. If failed, the user can select another computing methods available and iterate until a consistent result is obtained.

4 Computation methods

The initialization problem is generally defined as the solution of a nonlinear system of equations

$$0 = F(X)$$

where X is a vector composed of all the variables, derivatives of the differential variables (transformed into algebraic variables), and relaxed parameters (`fixed=false` or `weight=0`). The initialization GUI provides several computation methods for initialization. None are guaranteed to converge, so the user

should try sequentially several available methods until one works. This is possible because, if one method fails, another method can be tried using the last obtained result. In this section these methods as well as their characteristics will be presented.

4.1 Sundials

SUNDIALS[7] [5, 6] is a family of solvers which includes CVODE, for systems of ordinary differential equations, CVODES, variant of CVODE for sensitivity analysis, KINSOL, for systems of nonlinear algebraic equations, and IDA, for systems of differential-algebraic equations.
CVODE and IDA solvers have already been integrated in Scicos for dynamic simulation of models. IDA includes an optional user-callable module to recompute the initial conditions so as to be consistent with the given DAE system. This module uses inexact Newton methods with line-search strategies for accelerated convergence. We used this feature of IDA for initialization of models. IDA also permits inequality constraints to be imposed on the solution components. This can be used to implement max/min attributes used in Modelica.

4.2 Fsolve

Similar to IDA, Fsolve finds a zero of a system of nonlinear functions. This solver is based on the Modified Newton method and uses the QR factorization technique.

4.3 Optim

When the Modelica model is flattened, several variables exist for which start values are not given by the user. The compiler sets their start values to zero. It happens very often that the Jacobian matrix associated with the nonlinear equations in the very first step becomes singular, *e.g.,* consider the equation set (2) with $X_0 = [0,0]^t$ start values.

$$F(X): \begin{cases} 0 &= x-1 \\ 0 &= xy-1 \end{cases} \tag{2}$$

The first computed Jacobian matrix is singular and the Newton method fails. In order to overcome this problem, the user can choose the Optim solver. In this case, Scicos tried to obtain the solution by minimizing the cost function $C(X)$.

$$C(X) = \sum_{i=1}^{N} f_i^2(X) \tag{3}$$

The result of this optimization provides non-zero values that may be used as new start values for other methods, such as Sundials.

4.4 Homotopy

If Sundials or Fsolve methods fail, the user can use another methods often referred to as continuation or homotopy methods [7, 8]. Continuation methods are slower but more robust than Newton's method. In electrical circuit simulator, such as Spice and Spectre, in the case the Newton iteration fails, homotopy methods are used [9, 10, 11]. They start by modifying the model in such a way that the solution to the modified model is known or easy to compute, and such that a parameter controls the amount of the modification. Once the solution has been found for the modified model, the parameter is slowly returned to the original value, which causes the model set to return to its original form. As the parameter is changed, the solution is computed at each step, using the solution obtained from the previous step as the starting point. As long as the solution changes continuously as a function of the parameter, and the steps are small enough, the previous solution is very often a good starting point and the Newton's method converges. In other words, the model is written in the form $\phi(X,s) = 0$ where s is a real valued variable changing in the interval $[0,1]$. In Scicos, we use

$$\phi(X,s) = s\,F(X) + (1-s)\,(F(X) - F(X_0))$$

where X_0 is the initial starting point and $F(X)$ is the initialization equation to be solved.[8].

- for $s = 0$ the solution X_0 is known in advance or is easy to compute, *i.e.,* $\phi(X,s) = F(X) - F(X_0)$.
- for $s = 1$, the equation to be solved is the original initialization equation, *i.e.,* $\phi(X,s) = F(X)$
- the trajectory $X(s)$ is a continuous function of s.

This results in a contour curve between the solution of the initial system and the desired solution of the final system, as shown in Figure 3-A. The procedure for doing that is to slowly vary s from 0 to 1, computing the $X(s)$ at every step. In each step a the Newton method

[7] https://computation.llnl.gov/casc/sundials/

[8] $\phi(X,s) = s\,F(X) + (1-s)\,(X - X_0)$ is another popular form used in homotopy methods, but we could not get good results with.

is used to find the solution. The homotopy method is considerably slower than Newton's based method (Sundials & Fsolve).
The homotopy method may fail with some types of curves: simple discontinuities (see Figure 3-B), folds (see Figure 3-C), bifurcation (see Figure 3-D), and non-convergent curves (see Figure 3-E,F). Simple discontinuities are caused by discontinuities in the model equations. Folds, which are relatively common, occur when the solution curve doubles backs on itself and results in the model having multiple solutions for at least some values of $X(s)$.

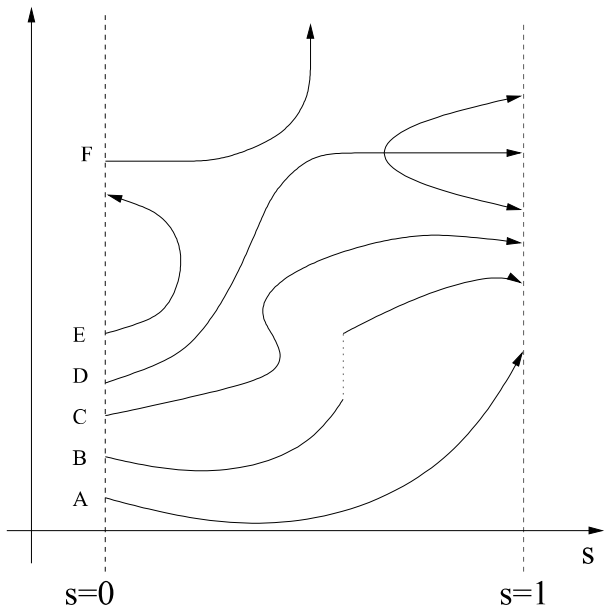

Figure 3: Homotopy curves

In the Spice and Spectre simulators, several variants of homotopy methods such as Gmin stepping and source stepping are used. In Gmin stepping, a resistor is placed between each node and ground. The resistor values are varied from zero to a very high value. In the source stepping method, the circuit sources are varied from zero to nominal values. In practice, source stepping trajectories are plagued by folds and so do not work very well. Gmin stepping is also subject to folds, but is much less susceptible to them that source stepping [9]. The Gmin stepping cannot be implemented without modifying the Modelica model, the source stepping method can, however, be employed.
There are several homotopy methods and software available. In Scicos, we use `HOMPACK`[9] package. This package provides several homotopy methods such as *fixed point*, *zero finding*, and *general homotopy curve tracking* using three different algorithms: ODE-based, normal flow, and augmented Jacobian [12, 13, 14]. We got better results with the *general homotopy curve tracking* method. `HOMPACK` can be successfully used for curves with folds, it fails however for discontinuous curves and curves with bifurcations.

[9] http://www.netlib.org/hompack/

4.5 Fsolve stepping

In this method, which is also an homotopy method, the s value is increased monotonically from 0 to 1 and in each step the nonlinear function $\phi(X,s)$ is solved using `Fsolve`. An interesting property of `Fsolve` is returning the absolute value of the solution when there are two complex conjugate solutions. For example, (4) has positive real valued solutions, but the solution curve $X(s)$ from $s=0$ to $s=1$ is not entirely in real domain, the solutions go to the complex domain and then returns.

$$F(X): \begin{cases} 0 = x+y-5 \\ 0 = xy+y-8 \end{cases} \tag{4}$$

The `Fsolve stepping` option does not support folds in the curve.

5 Example

Consider the electrical circuit shown in Figure 4. This circuit has been modeled with Modelica blocks in Scicos. The initialization GUI for this circuit is shown in Figure 2. For this electrical circuit, we would like to use the initialization GUI for two purposes: initialization from equilibrium state and for parameter sizing.

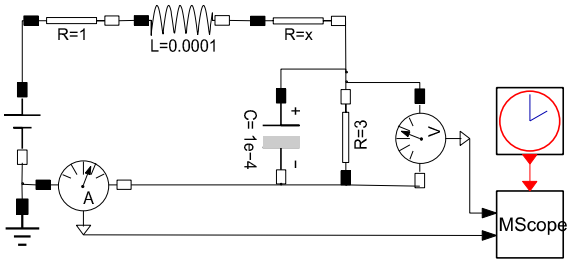

Figure 4: An electrical circuit to be initialized

- **Initialization:** In order to initialize from the steady state, all derivatives values should be fixed

to zero. For that, in the initialization GUI, derivatives of the differential variables which have been transformed into algebraic variables should be set to zero and their weight attribute should be set to one. In this circuit there are only two derivatives variables defined in inductor and capacitor blocks. In the initialization GUI, clicking on the name of these blocks, the user can change the attributes of derivatives. Then, the user can select a compute method and launch the initialization. Once the initialization finished, the obtained results will be displayed in the GUI and the user can start the dynamic simulation or retry another initialization method.

- **Parameter sizing:** In this case, the user needs to find a parameter value as a function of known system outputs. Suppose that the user needs to compute the resistance value of `Resistance1` that results in a current equal to two Amperes through the inductor at steady state. In this case, the `R` parameter in `Resistance1` should be relaxed by setting its weight attribute to zero. Then, the current through `Inductor1` should be fixed to two Amperes (by setting the weight attribute of `i` to one and its value to two). After selecting the compute method, the resistance value would be computed.

It is interesting to note that several initializations can be performed without modifying the Modelica program. Each initialization can be saved in an XML file and be used later.

6 Future works

The model initialization and model inversion which has been implemented asks some features:

- `weight:` At the current stage of the project, we focus on handling the cases `weight=0` and `weight=1` corresponding to `fixed=false` and `fixed=true`. At the next stage of the project, the user would be able to give the weight values between zero and one. In order to solve this optimization problem, we intend to implement several optimization methods such as least-square, quasi-Newton, conjugate-gradient, and Nelder-Mead (moving simplex) methods in Scicos.
- `initial equations:` They are not yet supported with `Translator`. It should be included in `Translator` by the end of the SIMPA2 project.
- `Arrays:` The way arrays and matrices are displayed in the initialization GUI should be improved. Currently, arrays are displayed element by element. When the number of elements of the array is large, this becomes inconvenient.
- `max, min, nominal`: Although implemented in the initialization GUI, they are not yet used in computing the initial values.
- `structural analysis:` In sizing or dynamic model inversion, when a fixed variable/parameter is relaxed, the user should fix a relaxed variable/parameter. The choice of variable/parameter is not straightforward and easy. In order to have a reasonable choice, a structural analysis should be performed. This task is a part of SIMPA2 project that should be implemented by 2009.
- Piecewise linear methods and Simplex are other methods that are used when the Newton based methods fail [15, 16, 17, 18, 19]. We intend to implement these methods in Scicos for model initialization.

7 Conclusion

In this paper the new initialization methodology of Scicos for initializing Modelica models has been presented. This methodology provides an easy and intuitive way for initializing Modelica models as well as model sizing and inverse problems.

References

[1] S. L. Campbell, J.P. Chancelier, R. Nikoukhah, Modeling and simulation in Scilab/Scicos, Springer Verlag publishing, 2005.

[2] J. P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, S. Steer, An introduction to Scilab, Springer Verlag, Le Chesnay, France, 2002.

[3] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM publication, Philadelphia, 1996.

[4] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, "Consistent initial condition calculation for differential-algebraic systems", *SIAM J. Sci. Comp.*, no. 19, 1998.

[5] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers," ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.

[6] A. C. Hindmarsh, "The PVODE and IDA Algorithms," LLNL technical report UCRL-ID-141558, December 2000.

[7] E.L. Allgower and K. Georg, *Numerical Continuation Methods, an Introduction*, Springer Ser. in Comput. Math. Springer-Verlag, Vol 13, 1990.

[8] E.L. Allgower and K.Georg, "Continuation and path following", *Acta Numerica*, 1993, pp. 1–64.

[9] Kenneth S. Kundert. “The designer’s guide to Spice and Spectre”. Kluwer academic publishers, 1995.

[10] A. Dyess, E. Chan, H. Hofmann, W. Horia, and Lj. Trajkovic, "Simple implementations of homotopy algorithms for finding dc solutions of nonlinear circuits", *Proc. IEEE Int. Symp. Circuits and Systems*, Orlando, FL, Vol. VI, 1999, pp. 290–293.

[11] W. Mathis, Lj. Trajkovic, M. Koch, and U. Feldmann, "Parameter embedding methods for finding dc operating points of transistor circuits", *Proc. NDES ’95*, Dublin, Ireland, 1995, pp. 147–150.

[12] L.T. Watson, Appl. Math. Comput. 5 (1979), pp. 297-311.

[13] L.T. Watson, "Golobally convergent homotopy algorithm for nonlinear systems of equations", *Nonlinear Dynamics*, Vol. 1, 1990, pp. 143–191.

[14] L. T. Watson, M. Sosonkina, R. C. Melville, A. P. Morgan, and H. F. Walker, "Algorithm 777: HOMPACK90: A suite of FORTRAN 90 codes for globally convergent homotopy algorithms", *ACM Trans. Math. Software*, Vol. 23, 1997, pp. 514–549.

[15] B. C. Eaves, "A Short Course in Solving Equations with PL Homotopies", *SIAM-AMS Proceedings* IX, 1976, pp. 73–143.

[16] K. Georg," An introduction to PL algorithms: Computational solution of nonlinear systems of equations", *Lectures in Applied Mathematics, American Mathematical Society*, 26, 1990, pp. 207–236.

[17] W.P.M.H. Heemels, J.M. Schumacher, and S. Weiland, "Linear complementarity systems", *SIAM Journal on Applied Mathematics*,Vol. 60 (2000), pp. 1234–1269.

[18] C.E. Lemke, and J.T. Howson, "Equilibrium points of bimatrix games", *SIAM J. Appl. Math.*, Vol. 12, 1964, pp. 413–423.

[19] D. M.W. Leenaerts, W.M.G. van Bokhoven. *Piecewise Linear Modelling and Analysis*. Kluwer Academic Publishers, Boston, 1998.

A XML file example

RCL_im.xml

```
<model>
   <name>RCL_im</name>
     <elements>
       <terminal>
          <name>R3</name>
          <kind>parameter</kind>
          <id>R3</id>
          <fixed value="true"/>
          <initial_value value="(5)"/>
          <weight value="1"/>
          <max value=""/>
          <min value=""/>
          <nominal_value value="1"/>
          <comment value="R3"/>
          <selected value="y"/>
          <kind_orig>fixed_parameter</kind_orig>
       </terminal>
      .....
  <struct>
     <name>CurrentSensor1</name>
       <subnodes>
        <terminal>
          <name>i</name>
          <kind>variable</kind>
          <id>CurrentSensor1.i</id>
          <fixed value="false"/>
          <initial_value value="(0)"/>
          <weight value="0"/>
          <max value=""/>
          <min value=""/>
          <nominal_value value="1"/>
          <comment value=""/>
          <selected value="y"/>
          <kind_orig>variable</kind_orig>
        </terminal>
       <struct>
     <name>p</name>
       <subnodes>
         .....

  </struct>
</elements>
  <equations>
    <equation value="0 = (‘VoltageSensor1.n.i‘ +  ...;"/>
    <equation value="0 = (‘Ground1.p.i‘ + ...;"/>
    <equation value="0 = (‘Resistor1.n.i‘ + ...;"/>
```

```
   ....
</equations>
<when_clauses/>
</model>
```