Robust Initialization of Differential-Algebraic Equations Using Homotopy

Michael Sielemann[1], Francesco Casella[2], Martin Otter[1], Christoph Clauß[3], Jonas Eborn[4], Sven Erik Mattsson[5], Hans Olsson[5]
[1]DLR Institute of Robotics and Mechatronics, Germany
[2]Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy
[3]Fraunhofer EAS, Dresden, Germany
[4]Modelon AB, Lund, Sweden
[5]Dassault Systèmes AB, Lund Sweden

Abstract

The new operator homotopy(..) was introduced in Modelica 3.2 to improve the solution of difficult initialization problems. The background and motivation for this approach is discussed and it is demonstrated how to apply it for mechanical, electrical and fluid systems. Furthermore, it is shown at hand of several examples how an inappropriate formulation might lead to ill-posed problems.

Keywords:
Initialization, DAE, homotopy, nonlinear equations

1 Introduction

A dynamic model describes how the state variables and thus the entire system behave over time. The state variables define the current condition of the model and have to be initialized when simulation starts. For this purpose, Modelica provides language constructs to define initial conditions such as initial equation sections (*Mattsson et. al., 2002*). The resulting constraints and all equations and algorithms that are utilized during the simulation form the initialization problem. Based on its solution, all variables, derivatives and pre-variables are assigned consistent values before the simulation starts.

Mathematically, the resulting problem is an initial value problem for a differential algebraic equation system (DAE) with dim(**f**) = *nx*+*nw* equations:

$$\mathbf{0} = \mathbf{f}\left(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, t\right), \quad \mathbf{x}(t) \in \mathbb{R}^{nx},\ \mathbf{w}(t) \in \mathbb{R}^{nw},\ t \in \mathbb{R}\ .$$

Here, **x** is the vector of state variables and **w** is the vector of algebraic unknowns. For simplicity of the discussion, we assume that the DAE has no hybrid part and is index-reduced, i.e. it has index 1, which means that the following expression is regular:

$$\begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial \dot{\mathbf{x}}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{w}} \end{bmatrix}.$$

Note, all the following results still hold for hybrid, higher index DAEs with small adaptations.

Initialization means to provide consistent initial values for $\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{w}_0$ so that the DAE is fulfilled at the initial time t_0. Since these are 2**nx*+*nw* unknowns and the DAE has *nx*+*nw* unknowns, additional *nx* equations must be provided which are called "initial equations" in Modelica:

$$\mathbf{0} = \mathbf{g}\left(\dot{\mathbf{x}}_0, \mathbf{x}_0, \mathbf{w}_0, t_0\right), \quad \dim(\mathbf{g}) = nx$$

The most often used initial equations are:

$$\dot{\mathbf{x}}_0 = \mathbf{0}$$

that is, steady-state initialization.

The result is usually a nonlinear system of algebraic equations, which has to be solved numerically. This does not always work right away for industrial problems as the commonly employed gradient-based local algorithms, such as the damped Newton method, provide local convergence only (even when using globalizations such as trust regions).

Modelica allows users to describe any model mathematically, which makes it highly flexible and powerful for simulation of heterogeneous multi-domain physical systems. However, this also means that no knowledge of the mathematical character of the problem equations can be introduced into the solver. Instead, an algorithm has to work on a general numerical problem (in contrast to domain-specific algorithms for nonlinear problems).

As a result, the success to solve initialization problems of state-of-the-art implementations of Modelica tools depends on the choice of iteration variables and the guess values for these variables defined with the start attribute. Library developers therefore typically implement approximate equations

in order to set these. They are usually formulated in terms of parameters (of e.g. the boundary conditions). At the same time, the iteration variables of the nonlinear equation systems may change after small topological modifications to the model. As a result it may become difficult for a library developer to provide a robust initialization capability.

Since a model becomes useless whenever initialization fails, and the current state-of-the-art is not fully satisfactory in this regard, we conclude that more reliable and robust methods are needed for a wider application of the Modelica modeling language by practitioners.

The goal of this contribution is to provide the solver with more information on the problem to solve. This is performed in an object-oriented way and seamlessly integrates with the concept of equation-based object-oriented languages.

2 Nonlinear Equation Solvers and Homotopy

The classic gradient-based iterative algorithms to solve nonlinear algebraic equation systems such as damped Newton's Method provide local convergence only, see, e.g., (*Dennis and Schnabel, 1996*), (*Deuflhard, 2004*), (*Kelley, 2003*). Such algorithms may fail due to various reasons such as the residuals not being Lipschitz continuously differentiable or containing local minima with respect to some norm introduced by the algorithm.

Several alternatives to these conventional methods exist. Homotopy is one of them and is considered in this contribution to meet the need for more robust initialization.

2.1 Established Homotopy Methods

In homotopy methods for solving nonlinear algebraic equation systems, the idea is to start with a simplified problem and continuously deform it to the difficult problem of interest. Even though this appears to be conceptually simple, several details of these methods and algorithms have to be considered. Unless certain prerequisites are met, the existence of the homotopy trace between the start and a solution, finite length of the path, nonexistence of singularities along the path and other important requirements are not guaranteed.

The homotopy is constructed from a system of residual equations that is easy to solve, as well as the one of interest, $\mathbf{F}(\mathbf{z}) = \mathbf{0}$. Here, a generic vector $\mathbf{z}$ of unknowns is used. In the Modelica case, this vector is: $\mathbf{z} = [\dot{\mathbf{x}}_0; \mathbf{x}_0; \mathbf{w}_0]$ and the equations are $\mathbf{F}$=[$\mathbf{f}$; $\mathbf{g}$]. The homotopy is then a system of equations with one higher dimension and is denoted by

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \mathbf{0}.$$

The additional dimension is the homotopy or continuation parameter λ. It is typically restricted to the range $0 \le \lambda \le 1$ such that $\boldsymbol{\rho}(\mathbf{z}, 0) = \mathbf{0}$ is solved easily and $\boldsymbol{\rho}(\mathbf{z}, 1) = \mathbf{F}(\mathbf{z})$ is the system of interest.

At least three different homotopies are discussed in literature. We introduce the *Fixed Point Homotopy* following (*Chow et. al., 1978*) as

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot (\mathbf{z} - \mathbf{z}_0).$$

Here, $\mathbf{z}_0$ is the start iterate. According to (*Keller, 1978*), the *Newton Homotopy* (or Global Homotopy) is defined as follows:

$$\begin{aligned} \boldsymbol{\rho}(\mathbf{z}, \lambda) &= \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot (\mathbf{F}(\mathbf{z}) - \mathbf{F}(\mathbf{z}_0)) \\ &= \mathbf{F}(\mathbf{z}) - (1 - \lambda) \cdot \mathbf{F}(\mathbf{z}_0) \end{aligned}$$

Finally, the *Affine Homotopy* is introduced following (*Wayburn and Seader, 1987*) as

$$\boldsymbol{\rho}(\mathbf{z}, \lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1 - \lambda) \cdot \mathbf{F}'(\mathbf{z}_0) \cdot (\mathbf{z} - \mathbf{z}_0)$$

Here, $\mathbf{F}'(\mathbf{z}_0)$ denotes the Jacobian of the residual equations at the start iterate.

The Newton Homotopy has the advantage of scale-invariance (*Wayburn and Seader, 1987*). However, the simple problem $\boldsymbol{\rho}(\mathbf{z}, 0) = 0$ may have several solutions and infinite loops that do not cross $\lambda = 1$ may result. Such tracks are called isolae (*Choi and Book, 1991*). The Fixed Point and Affine Homotopies only contain a single solution to the simple problem. Therefore, starting continuation inside an isola is impossible. The Affine Homotopy is also scale-invariant and the Fixed Point Homotopy is not (*Wayburn and Seader, 1987*).

Affine and Fixed Point Homotopies in turn may prescribe traces, which diverge toward an infinite value of some elements of the unknowns $\mathbf{z}$. Obviously, such traces cannot be followed numerically as the arc length is infinite and because the sign may change.

We note that (*Chow et.al., 1978*) provides theorems on the success of the Fixed Point Homotopy with probability one in the sense of a Lebesgue measure. Success means that the track is of finite length, bounded and free of singularities (with the exception of turning points, which are not critical). The associated coercivity conditions on the residual equations were successfully employed in the area of analog circuit simulators for example. However, it is

neither possible to translate the boundedness condition on the solution of the residual equations **F**(**z**) nor the Inner Product Condition on **F**(**z**) to general multi-domain physical modeling as needed by a Modelica model. The former is the case due to the existence of unsaturated amplifier components, which arise in several applications, and the latter due to the lack of energy dissipation in component models to compensate the effect of boundary conditions in several physical domains other than electronics.

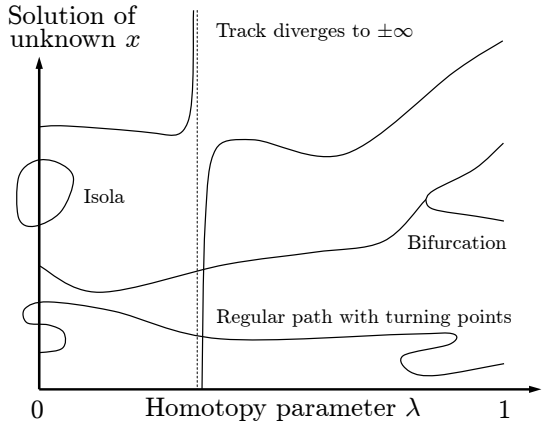

Figure 1: Illustration of failure modes of homotopy

In summary, the experience of the authors shows that such established homotopy methods are not sufficiently robust due to be above mentioned failure modes. Furthermore, an implementation of the Fixed Point Homotopy within Dymola 7, which was available since several years, did not provide indications of increased robustness of this approach with respect to the Newton solver in practical applications.

2.2 General Problem-Specific Homotopy

The issues in the general homotopies introduced so far stem from continuously deforming two rather unrelated systems of equations into each other. In the case of the Fixed Point Homotopy the simplified problem is the linear system $\mathbf{z} - \mathbf{z}_0$.

The source of the problem is the "lack of additional information" that can be utilized for the solution. In order to improve this situation for Modelica, a problem-specific homotopy is introduced:

- By deriving the simplified system from the actual system of interest, and
- by formulating the simplified system such that a homotopy to the actual problem of interest be free of singularities.

The formulation of the simplified system is problem-specific and allows modelers to infuse their knowledge about the physics of the problem into the way the equation system is solved (cf. Introduction). The approach is compatible with object-orientation and declarative modeling and is understood as something introduced by domain experts to selected key equations. The goal is the formulation:

$$\boldsymbol{\rho}(\mathbf{z},\lambda) = \lambda \cdot \mathbf{F}(\mathbf{z}) + (1-\lambda) \cdot \tilde{\mathbf{F}}(\mathbf{z}).$$

Here, $\mathbf{F}(\mathbf{z})$ is the actual problem and $\tilde{\mathbf{F}}(\mathbf{z})$ is the simplified one. Based on a proposal by M. Otter, M. Sielemann and F. Casella, the new built-in operator, `homotopy(..)` was introduced in Modelica 3.2. It depends on two arguments, namely `actual`, the Real expression describing the actual problem, and `simplified`, the Real expression corresponding to the simplified problem. The Modelica translator can then expand this operator according to the homotopy. For the homotopy given above, which will be used throughout the remaining part of this article, the expression

```
homotopy(expr1, expr2)
```

is thus expanded to

$$\lambda \cdot \mathrm{expr1} + (1-\lambda) \cdot \mathrm{expr2}.$$

In contrast to other language constructs, the benefit of using this operator is that only one equation system for any number of steps is needed for initialization, and that it is logically defined how to transform one equation system into the other.

3 Implementation in Modelica Tools

The implementation of the new homotopy operator in a Modelica tool is rather straightforward: During the symbolic manipulation phase (BLT transformation, Pantelides algorithm etc.), the operator is treated as a function with two arguments. When generating code, the tool has to conceptually perform one homotopy iteration over the whole model and not several homotopy iterations over the respective local algebraic equation systems. The reason is that the following structure can be present:

$$\mathbf{w}_1 = \mathbf{f}_1(\mathbf{x}) \quad \text{// has homotopy operator}$$
$$\mathbf{0} = \mathbf{f}_2(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}_1, \mathbf{w}_2)$$

Here, a local non-linear equation system $\mathbf{f}_2$ is present. The homotopy operator is, however, used on a variable that is an "input" to the non-linear algebraic equation system and modifies the characteristics of it. The only useful way is to perform the homotopy iteration over $\mathbf{f}_1$ and $\mathbf{f}_2$ together.

This approach is "conceptual", because more efficient implementations are possible, e.g. by determining the smallest iteration loop, that contains the equations of the first BLT block in which a homotopy operator is present and all equations up to the last BLT block that describes an equation system.

Various continuation algorithms have been suggested in literature, which are all suitable to trace homotopies of the type considered herein (e.g. pseudo arc-length algorithms). Popular examples are Hompack (*Hompack, 2010*) and Alcon2 (*Elib, 2010; Deuflhard et. al. 1987*).

In order to validate the methodology, a test implementation was developed by M. Sielemann, which utilizes the Dymola software (*Dymola 2010*) and the Loca continuation algorithms of Trilinos (*Heroux et. al., 2005*). One practical advantage of Loca over Hompack and Alcon2 is that the sensitivities of the homotopy with respect to the continuation parameter λ do not have to be provided. For Hompack and Alcon2 this has to be provided and had to be implemented using finite differences. The Loca/Dymola implementation has the following features:

- It provides three options for the treatment of the suggested homotopy operator. Normally, it is expanded to the given homotopy expression. Alternatively, simplified equation sets are obtained by inlining either argument. In case of the simplified argument, maximum structural simplifications of the equation system result.
- The user is able to manually prescribe whether to use homotopy initialization or not. This is an important feature for library development and debugging, and may be useful for users, too (e.g. if a local gradient based solver converges to a mathematically valid, but physically unreasonable solution or when a local gradient based solver does not converge and a user does not want to wait at the start of each simulation until the software realizes this and switches to homotopy initialization).
- Verbose information on the homotopy is optionally provided, which is useful for library development and debugging. In particular, the homotopy traces are visualized. Like this, it is possible to reconstruct what happens during the solution of the simplified problems and the homotopy transformation.

Additionally, Dymola 7.5 Beta also supports the homotopy operator. It was used for some of the application examples.

4 Application Examples

In this section several examples are given how to utilize the homotopy operator in different physical domains in order to solve difficult initialization problems.

4.1 Mechanical Systems with Kinematic Loops

Whenever kinematic loops are part of a mechanical system, non-linear algebraic equation systems are present. If these equation systems are solved numerically, the user has to provide guess values for the iteration variables in order that the system can be initialized. The issues are first demonstrated at hand of a simple example, the four bar mechanism, see Figure 2:

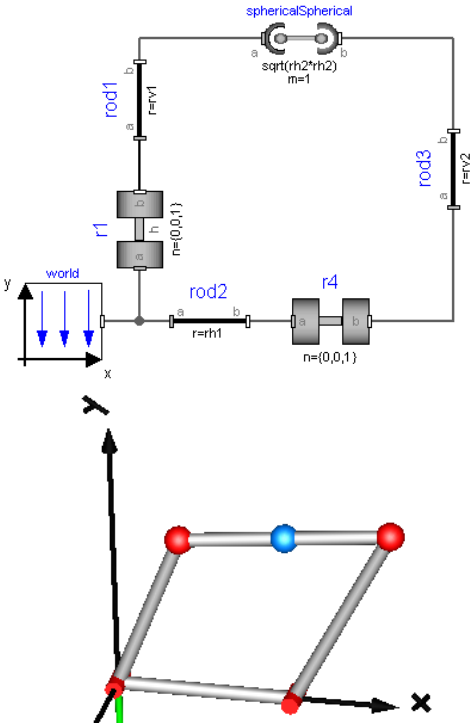

Figure 2: Four bar mechanism (top: Modelica model, bottom: animation).

The four bar mechanism consists of 4 connected revolute joints where the rotation axes of the joints are all parallel to each other. Since this mechanism is over constrained (e.g., the forces perpendicular to the kinematic loop cannot be uniquely determined), the upper two revolute joints are replaced by spherical joints which gives the same kinematic motion, but all quantities can be uniquely computed. Since joint r1 shall be driven by a drive train, the angle of this joint, "r1.phi" and its derivative are defined to be states by selecting in the "Advanced" menu of joint "r1" the option "stateSelect = StateSelect.always".

This mechanical system gives rise to 9 nonlinear algebraic equations that are transformed by Dymola to one non-linear algebraic equation in one unknown. This equation is the constraint that the distance between the two spherical joints is constant. Formally, this nonlinear algebraic equation has the form:

$$0 = f(r1.phi, r4.phi)$$

where r1.phi is the "known" state and "r4.phi" is the angle of the right lower revolute joint that is used as

iteration variable. This nonlinear equation has two solutions that correspond to the two configurations of the mechanism. In order to initialize this mechanism, a "guess" value for variable r4.phi has to be provided.

It is always a useful strategy to define a mechanism in a reference configuration in which all generalized joint coordinates are zero and where all relevant kinematic quantities can be easily determined. When the mechanism is initialized in this way, the nonlinear equations of the initialization problem are fulfilled. In the case of the four bar mechanism, the selected reference configuration (in which r1.phi = r4.phi = 0) is selected such that the left bar is directed along the y-axis and the lower bar along the x-axis, respectively (see left part of Figure 3 below).

Problems arise, if the simulation of the mechanism shall not start in the reference configuration, but at a user-defined angle r1.phi = phi0. Depending on the "guess" value of "r4.phi" the numerical solver might no longer find a solution, or if it computes a solution, it might be the wrong configuration.

In the example of Figure 3, a guess value of r4.phi = 45° is selected and r1.phi is changed from r1.phi = 0°, in steps to -20°. The initial solutions found by Dymola are shown in Figure 3:

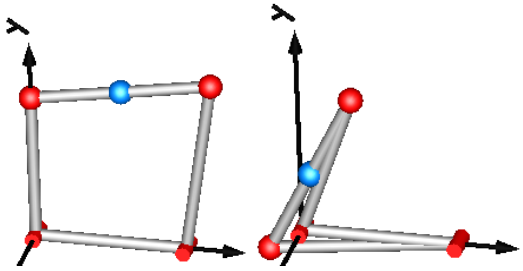

Figure 3: Initial solutions:
(left: r1.phi = 0°, right: r1.phi = -20°)

Starting at about 18.9° the configuration is changing to an undesired configuration. This type of initialization is not robust, since for every change of the initial states, all guess values need to be properly adapted, which is usually difficult (not practical) if the system is no longer in its reference configuration.

The homotopy operator opens up a completely new direction: In the model of the revolute joint, the equation for the joint angle is changed to

```
if homotopyInitialization then
  angle = phi_offset + homotopy(phi,0);
else
  angle = phi_offset + phi;
end if;
```

where `homotopyInitialization` is a Boolean parameter that is set to **true** for r1 and set to **false** for r4. Furthermore, the start value of r4.phi = 0 (the value from the reference configuration). The meaning is that independently which start value is given for r1.phi, the mechanism is initialized in its reference configuration r1.phi = 0 (where the nonlinear algebraic equation is identically fulfilled) and then r1.phi is moved by the homotopy method until it reaches its start value. In every iteration a good guess value exists from the previous step and therefore the nonlinear equation is solved and remains in the configuration of the reference configuration. As a result, a very robust initialization of the mechanism is obtained, see Figure 4:

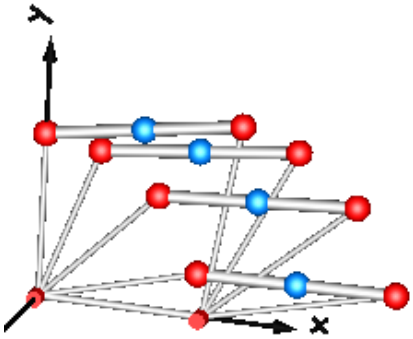

Figure 4: Initial solutions for
r1.phi.start = 0°, -20°, -45°, -75°

The four bar mechanism was only introduced to demonstrate the issues on a simple mechanism[1].

The sketched initialization technique shall now be applied on a much more involved example: A "**Delta**" robot (*Clavel 1990*). This robot is commercially available by several companies, e.g., by ABB under the name "FlexPicker™"[2]. A suitable reference configuration of this robot is shown in Figure 5:

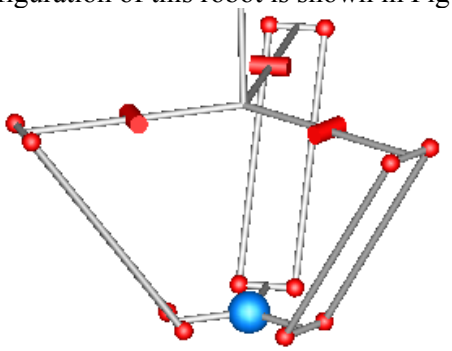

Figure 5: Delta robot in its reference configuration.

At the top, the robot consists of 3 actuated revolute joints that each drives a parallelogram. Every parallelogram consists of 4 spherical joints. In the bottom, the three parallelograms are rigidly mounted on a plate (in the figure visualized by a blue sphere that marks the center of mass of the load body that is attached to this point; in commercial robots, there is

[1] The equation system can be solved analytically when using an Assemblies.JointRRR joint from the Modelica.Mechanics.MultiBody library.

[2] FlexPicker is a trademark of ABB.

usually an additional revolute joint here). Overall, this robot has 3 revolute joints, 12 spherical joints and has 6 coupled kinematic loops. The robot has 3 degrees of freedom and can be controlled by the 3 revolute joints. By construction, the load plate is always parallel to the mounting plate on top, independently of the actual joint angles. Within its workspace, the robot can move very fast to a desired position. Since the motors that are mounted on the top plate are not moved, accelerations can be up to 30 g and speeds of 10 m/s can be reached.

Both direct kinematics (= given the joint angles, compute the position of the load), as well as the inverse kinematics (= given the load position, compute the joint angles) give rise to nonlinear algebraic equation systems. The more complicated case is the direct kinematic solution. When the robot is built up with "Joints.SphericalSpherical" joints (that each introduces a length constraint between two spherical joints), then Dymola transforms the system of 87 nonlinear algebraic equations down to 6 equations. If the joint angles are given, the resulting equation system has 16 configurations, but only the one shown in Figure 5 is the desired one. With the homotopy initialization, this system is initialized in the following way:

1. In the reference configuration, the absolute position r[3] of the center point of the load plate, as well as the rotation angles phi[3] from the inertial frame to the load frame can be easily analytically computed ($r = \{0, 0, -\mathrm{sqrt}(L^2 - (r_1+r_2-r_3)^2)\}$, phi = {0,0,0}). These values are provided as start values to the load body (since Dymola selects them as iteration variables of the nonlinear equation system).
2. The homotopy initialization of the revolute joints is switched on. So, for given start angles, the robot always starts first in the reference configuration and then moves the angles to the desired start configuration.

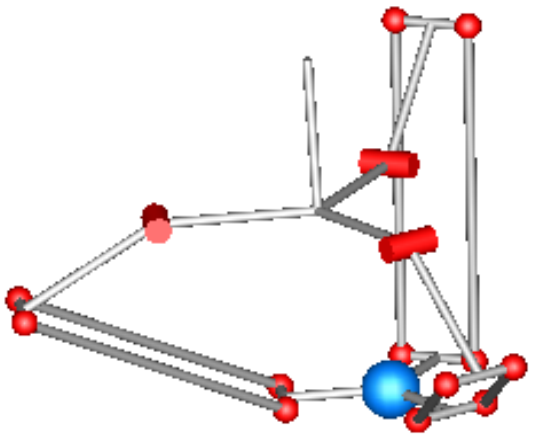

Figure 6: Delta robot initialized in configuration {45°, -45°, 30°}.

Practical experience shows that within the technical workspace of this robot, the initialization is very robust. A typical example is shown in Figure 6.

The path of the three position variables of the load mass as function of the homotopy parameter (computed with Loca) is shown in Figure 7. As can be seen, the three paths are nearly linear and therefore even simple homotopy methods (like fixed step methods) will work.

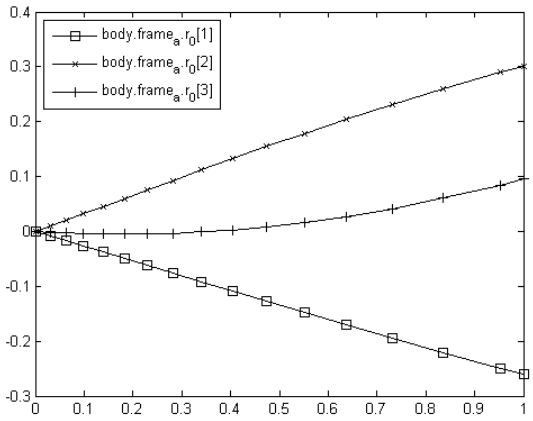

Figure 7: Homotopy path of the absolute position variables for the initialization of Figure 6.

4.2 Analog Electronic Circuit

In electronic circuits, operation starts often after the power supply is switched on. Power supply is in most cases a constant operating voltage of 15V, 5V or others, often a split supply with +15V and -15V is used. After switching on power supply, an initial value of all variables (voltages and currents) is reached, especially capacitors are loaded. The state in which no variable is varying any more is called DC (direct current) operating point. Its calculation is often a challenge for which homotopy operators are useful.

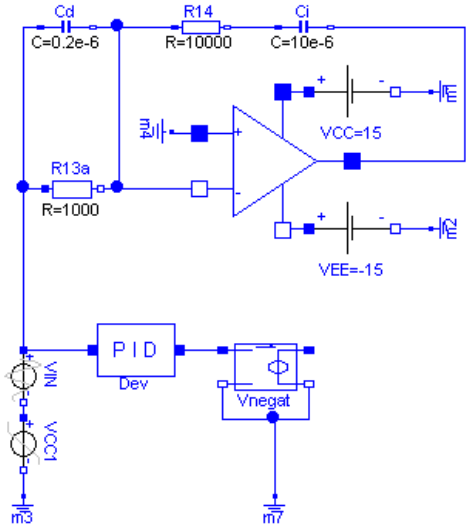

Figure 8: PID controller circuit with µA741 operational amplifier.

Figure 8 shows a simple PID controller circuit (*Tietze and Schenk, 2002*) using a µA741 operational amplifier model (*Horowitz and Hill, 1989*) which is composed of 21 NPN and PNP transistors of the Modelica Standard Library, see Figure 9:

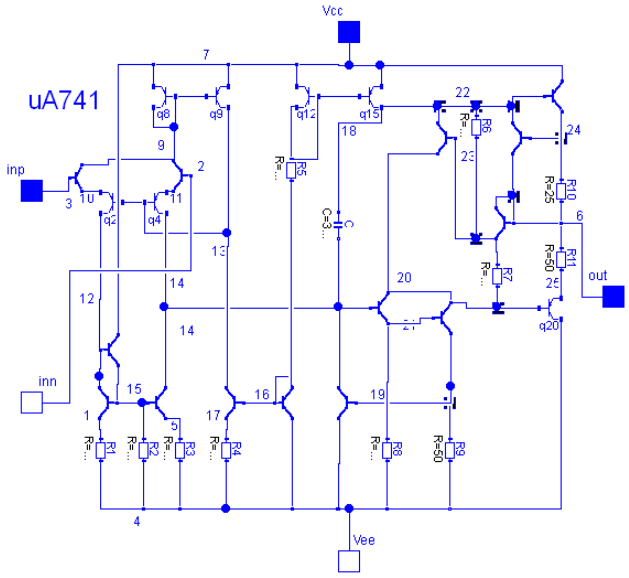

Figure 9: Operational amplifier µA741 composed of 21 NPN and PNP transistors.

To compare the due power supply (VCC 15V, VEE -15V) limited controller output with the ideal unlimited behaviour, a mathematical PID controller model is inserted in parallel. Typical simulation results with a comparison of the two models are shown in Figure 10. Translating this circuit, results in a system of 240 nonlinear algebraic equations that is reduced by Dymola to a set of 38 nonlinear algebraic equations that have to be solved during initialization (during simulation, only a system of 17 linear equations is present). With Dymola 7.4 (and most likely also with any other Modelica tool), initialization of this circuit fails, i.e., the DC operating point cannot be calculated.

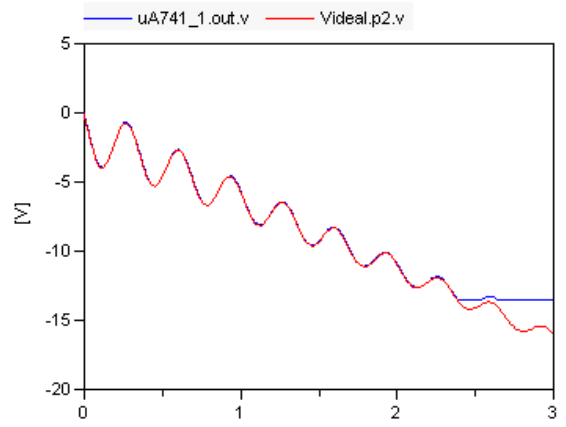

Figure 10: Comparison of circuit PID with the ideal mathematical PID controller.

A successful initialization is possible by replacing the constant supply sources VCC and VEE by ramp sources which start at zero, followed by a transient simulation until all variables remain constant. In general, this way is cumbersome and error prone since the circuit has to be changed manually. Furthermore, the ramping up during a simulation introduces oscillations and simulation has to be long enough until the vibrations "died out".

The situation changes completely, if the homotopy operator is used by changing the constant voltage model according to

```
model ConstantVoltage_Homotopy
  import Modelica.Electrical.Analog;
  extends Analog.Interfaces.OnePort;
  parameter Modelica.SIunits.Voltage V;
equation
  v = homotopy(V,0.0);
end ConstantVoltage_Homotopy;
```

This definition starts the constant voltage at zero and during homotopy initialization it is ramped up to the desired voltage V. During the ramping, all derivatives are zero and therefore it is a ramping along steady-states. Simple homotopy algorithms fail in this case. In this example, the Loca algorithm was used to calculate the homotopy initialization. In Figure 11 the non-trivial variation of an internal voltage of the operational amplifier is shown with respect to the homotopy variable λ changing from zero to one. Due to the sharp edge at $\lambda = 0.18$, a homotopy method with a variable step size is needed in this case.

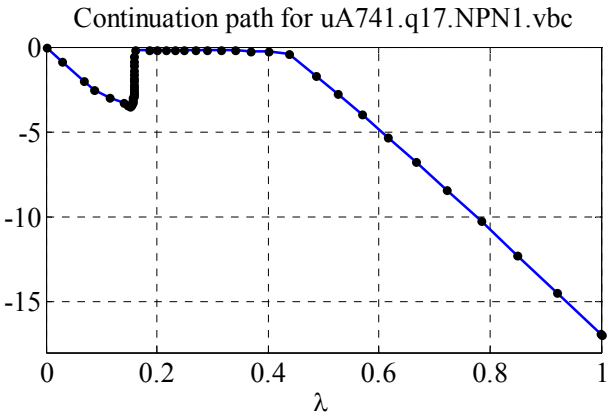

Figure 11: Homotopy path for a voltage variable of the operational amplifier with respect to λ.

4.3 Hydraulic Networks

Hydraulic networks are typically characterized by the simultaneous presence of components with large and small pressure losses, by mixing points, and by nonlinear momentum balance equations, which depend on the fluid properties, e.g., the density. As a

result, the system of nonlinear equations during steady-state initialization is typically large and strongly nonlinear. Their numerical solution is therefore problematic, unless relatively accurate start values are set for the iteration variables.

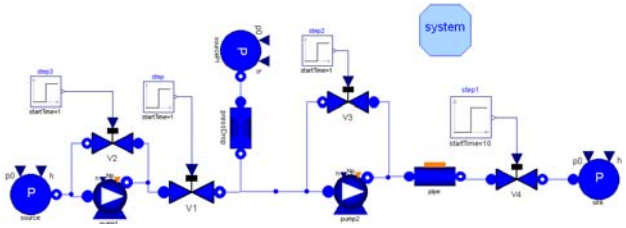

Figure 12: A hydraulic network.

An example case built with the ThermoPower 3 library is shown in Figure 12. A pump with a recirculation and a control valve sucks fluid from a low pressure source. The fluid is then mixed with the flow coming from a second intermediate pressure source through a short pipe, and further pumped through a long pipe (with mass and energy storage) into a high-pressure sink. The pressure losses in the two pipes are small, compared to the pressure losses across the valves and pumps.

The resulting initialization problem has 13 iteration variables after tearing, among which two flow rates and four pressures. If the start values of those six variables are not accurately set, the standard nonlinear solver in Dymola fails to converge.

This initialization problem can be made much easier to solve by substituting the original momentum balance equations in the pump and pipe models by linear, constant-coefficient ones, which are tuned based on nominal operating data, and then by applying the homotopy transformation to bring the model back to its original form.

More specifically, the pressure losses in the short and long pipes are computed by linear m_{flow}-dp relationships, passing through the origin and through the nominal flow and nominal pressure loss point (these data must be provided as parameters). In the case of the pump, the tangent to the flow-head curve at the nominal flow rate is used instead of the original curve. By the simple substitutions of these two equations, the hydraulic problem becomes linear (two linear systems with five and three unknowns), while all the enthalpies and fluid properties are calculated by simple assignments once the flow rates are known. As a consequence, *no start value at all* is required to guarantee convergence of the simplified problem; the homotopy transformation then solves the original nonlinear problem without further intervention by the end user.

It is interesting to note that the homotopy paths of the iteration variables are smooth and do not show any kind of singularity or turning point even if the actual steady state has a substantial mismatch with the nominal data used to set up the simplified model. As an example, Figure 13 shows the continuation paths for two pressures and two flows if the valve V4 on the far right is closed by 90% at initialization, thus reducing all flows in the circuit to a small fraction of the nominal flow.

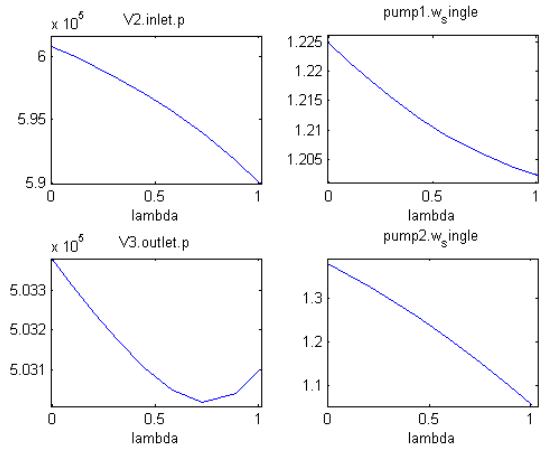

Figure 13: Homotopy paths of 4 iteration variables.

4.4 Calibration of A/C Heat Exchanger

A typical problem in air conditioning system and component design is to calibrate a heat exchanger model to measurement data. This is performed using steady-state initialization in a test bench with given boundary conditions, like the one shown in Figure 14.

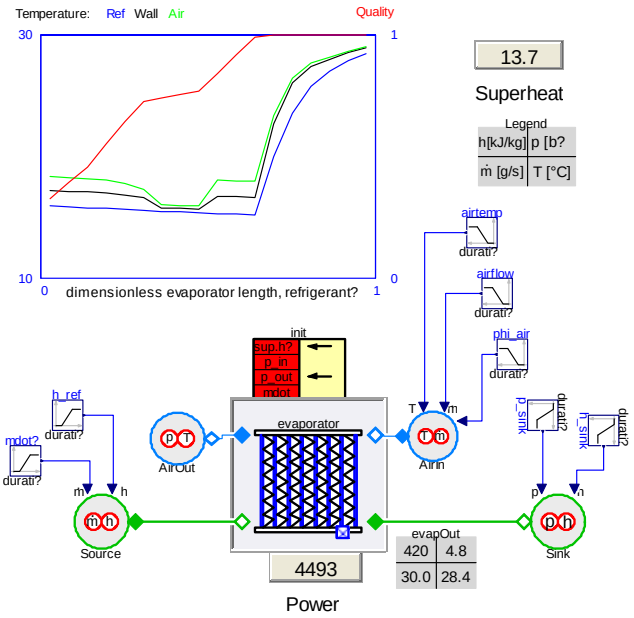

Figure 14: Evaporator calibration test bench, from the AirConditioning library.

Heat transfer on the air side can be correlated using the Nusselt number, $Nu = k_c D_{hyd}/\lambda_F$, which relates the heat transfer coefficient k_c to the hydraulic di-

ameter D_{hyd} and the fluid thermal conductivity λ_F. Normally, the calibration can be performed during initialization by solving for Nu as an unknown parameter using initial equations.

```
  parameter Modelica.SIunits.NusseltNumber
    Nu_air(fixed=false, start=10)
                  "global Nusselt number";
initial equation
  hex.summary.Qdot_air = P_measured;
```

In most cases this solves perfectly fine using standard methods, and can be combined to calibrate several parameters simultaneously, for example both heat transfer and pressure drop. But sometimes it is difficult to reach the desired solution, P_measured, because it is close to the maximum cooling capacity. The solver will then fail to converge.

Using the homotopy approach, the Nu-number may be used as a control signal, starting at a given value for which the steady-state initialization converges, see the code below:

```
  parameter Modelica.SIunits.NusseltNumber
    Nu_air(fixed=false, start=10)
    "global Nusselt number";
  parameter Modelica.SIunits.NusseltNumber
    Nu_start=10 "starting  Nusselt number";
initial equation
  0 = homotopy(
actual = hex.summary.Qdot_air - P_measured,
simplified = Nu_air - Nu_start);
```

The path that the homotopy solver takes can be illustrated with a plot of Qdot_air vs. Nu, see Figure 15. The starting value is taken in the middle of the sloping curve, and the solver will then converge to the desired solution, if one exists. This method has been used to calibrate over large sets of data with excellent results.

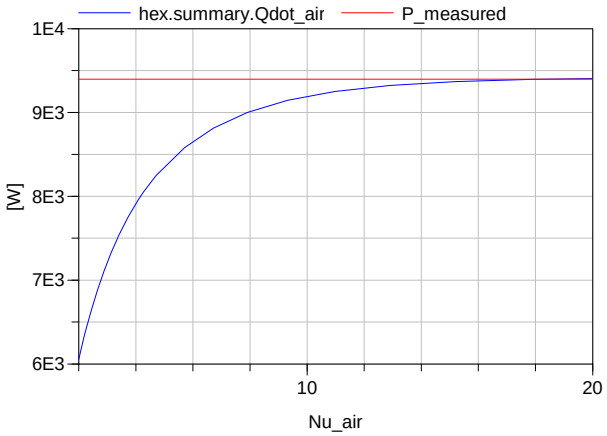

Figure 15: Steady-state performance of heat exchanger as function of Nu-number on the air side. The solution Nu_air = 19.7 is very close to maximum Qdot=9420 W.

5 Ill-posed Examples

Unfortunately, it is quite easy to formulate ill-posed problems with the homotopy operator, so that initialization will fail. Below, a number of simple examples are given to demonstrate different kinds of issues.

5.1 Singular Simplified System

The "simplified" problem in the homotopy formulation might be formulated too simple by removing all dependencies of a variable, as shown in the next example:

```
x + 2*homotopy(y,1) = 5
2*x - homotopy(y,1) = 0
```

Note, the "simplified" problem is actually:

```
x + 2   = 5
2*x - 1 = 0
```

and this equation system does not have a solution although the "actual" problem has a solution. There are different variants of this type of problem. For example, the "simplified" system might remove variables that are used as iteration variables in a system of equations and then the system is singular, although a different selection of iteration variables might make the system regular.

Since such cases can easily appear, the minimum requirement is that a tool reports these problems during translation. Conceptually this is easy, by performing an assignment for the "simplified" problem which would fail (with good diagnostics), if this problem is structurally singular.

A tool might also perform a more involved treatment:

1. For the tearing algorithm, select only iteration variables, that are appearing in the "actual" and in the "simplified" problem formulation (does not work for the problem above).
2. Solve simplified problem with symbolic manipulations (does not work for the problem above).
3. Remove the homotopy operator from certain equations, until the "simplified" system is structurally regular. This would work in the example above, e.g., by removing the homotopy operator from the second equation.
4. The homotopy formulation of appropriate equations is changed. In the example above, one can observe that the modeler defined with the second equation that "y" shall be used for the "actual" problem and "1" for the simplified" problem, i.e., the modeler defined "y=1" for the "simplified" problem. This information

allows to rewrite the second equation to:

```
homotopy(2*x,1) - y = 0
```

which results in a regular "simplified" system.

5.2 Singular Intermediate System

Singular systems might also occur for a combination of the "simplified" and "actual" problem formulation, i.e., when $0 < \lambda < 1$. A typical example is the following where homotopy moves from an "initial state" to a "steady state" formulation (i.e., using Fixed Point Homotopy):

```
model DoNotUse
  Real x;
  parameter Real x0 = 0;
equation
  der(x) = 1-x;
initial equation
  0 = homotopy(der(x), x - x0);
end DoNotUse;
```

After the initial equation is expanded to

$$0 = \lambda \cdot \dot{x} + (1-\lambda)\cdot(x - x_0)$$

the two equations can be solved for the unknown x by eliminating the derivative of x:

$$x = \frac{(1+x_0)\lambda - x_0}{2\lambda - 1}$$

This equation has a singularity at $\lambda = 0.5$, see Figure 16. A homotopy solver will usually not be able to compute the solution and therefore initialization will fail.

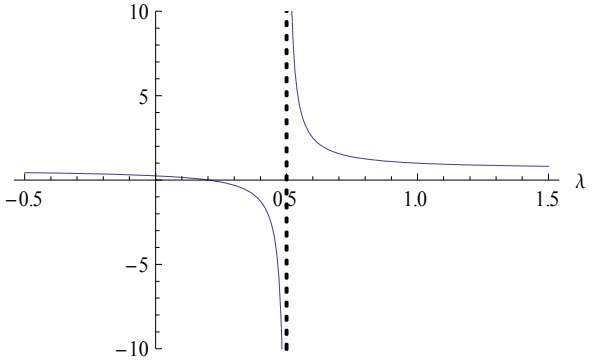

Figure 16: Solution to ill-posed example for $x_0 = 0.25$

5.3 Bifurcation of Intermediate System

Ramping of boundary conditions is a straightforward way to employ homotopy. Some care has to be taken however when using this pattern, which is illustrated for a flip flop, see the simple analog electric circuit of Figure 17:

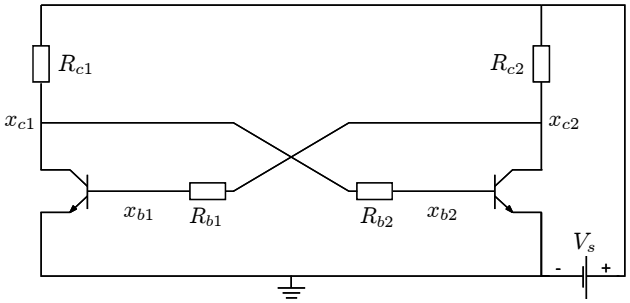

Figure 17: Flip-flop circuit leading to several solutions during the homotopy iteration.

This flip flop circuit has three steady state solutions out of which two are stable. If a homotopy is constructed by ramping up the source voltage V_s, then a bifurcation will show up in the homotopy track. This bifurcation shows up at the point at which the base-emitter junction of the transistor is triggered and the three steady state solutions emerge. In non-trivial applications, such bifurcations are numerically difficult to handle and shall thus be avoided under any circumstances. The following figure illustrates the homotopy trace that results in such a natural parameter continuation strategy (also called source stepping). Here, a simple Ebers-Moll transistor model was used.

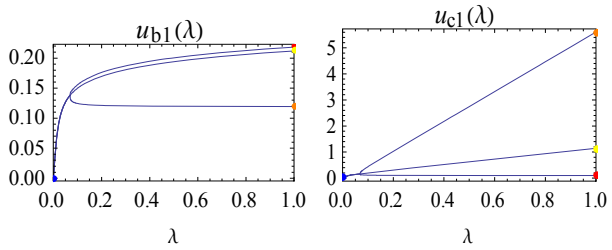

Figure 18: Voltages at base and collector of transistor 1 in flip-flop circuit. At λ = 0.15 a bifurcation to three solutions occurs.

6 Conclusions

The homotopy operator introduced in to the Modelica language in version 3.2 (*Modelica 2010*) opens up completely new possibilities to robustly initialize Modelica models. Several examples have been given to demonstrate the usage in different domains. Additionally, large power plant applications with up to 671 iteration variables for steady-state initialization are discussed in (*Casella et. al. 2011*). Due to the successful applications, it is planned to introduce this operator at appropriate places in to the next version of the Modelica Standard Library, in order to improve the initialization of Modelica user models.

As demonstrated by several examples in section 5, it is easy to misuse the homotopy operator resulting in failed initialization. As a "rule of thumb", the

homotopy formulation should not change the "structure" of the equation system, i.e., it should be based on the simplification of terms, but not by solving a completely different problem (e.g. moving from a simplified system that is initialized at given states to a steady-state formulation might easily fail, see section 5.2). Furthermore, it is always useful to inspect how the start-up of the "real" system works and mimic this "start-up" with the homotopy formulation, if this is possible.

There is still room for improving initialization. One issue is that still guess values might be needed for iteration variables (see, e.g., the Delta robot in section 4.1) and the iteration variables are selected by the tool. One remedy might be to introduce an additional enumeration attribute for variables, such as, "iterationSelect" that allows a library developer to directly suggest useful iteration variables with the enumeration values "never, avoid, default, prefer, always", in a similar way as for the existing attribute "stateSelect" to guide the state selection.

7 Acknowledgements

Partial financial support of DLR and Fraunhofer by BMBF (Förderkennzeichen: 01IS07022F) for this work within the ITEA2 project EUROSYSLIB (www.eurosyslib.com; funding number 06020) is highly appreciated.

References

Allgower E.L., Georg K. (2003): **Introduction to numerical continuation methods**. SIAM Classics in Applied Mathematics.

Casella F., Sielemann M., Savoldelli L. (2011): **Steady-state initialization of object-oriented thermo-fluid models by homotopy methods**. Modelica'2011 Conference, Dresden, March 20-22.

Choi S.H., Book N.L. (1991): **Unreachable roots for global homotopy continuation methods**. AIChE Journal 37, pp. 1093-1095.

Chow S.N., Mallet-Paret J., Yorke J.A. (1978): **Finding Zeroes of Maps: Homotopy Methods That are Constructive With Probability One**. Mathematics of Computation 32, pp. 887-899.

Clavel R. (1990): **Device for the Movement and Positioning of an Element in Space**. US Patent No. 4,976,582, December 11, 1990. Download: http://v3.espacenet.com/publicationDetails/biblio?CC=US&NR=4976582&KC=&FT=E

Elib (2010): http://elib.zib.de/pub/elib/codelib/alcon2/. Accessed November 2010.

Dennis J.E., Schnabel R.B. (1996): **Numerical methods for unconstrained optimization and nonlinear equations**. SIAM Classics in Applied Mathematics.

Deuflhard P., Fiedler B., Kunkel P. (1987): **Efficient numerical path following beyond critical points**. SIAM Journal on Numerical Analysis, Society for Industrial and Applied Mathematics, 24, 912-927.

Deuflhard P. (2004): **Newton Methods for Nonlinear Problems**. Affine Invariance and Adaptive Algorithms. Springer.

Dymola (2010): **Dymola 7.4**. http://www.3ds.com/products/catia/portfolio/dymola

Heroux M.A., Bartlett R.A., Howle V.E., Hoekstra R.J., Hu J.J., Kolda T.G., Lehoucq R.B., Long K.R., Pawlowski R.P., Phipps E.T., Salinger A.G., Thornquist H.K., Tuminaro R.S., Willenbring J.M., Williams A., Stanley K.S. (2005): **An overview of the Trilinos project**. ACM Transactions on Mathematical Software, 31, 397-423.

Hompack (2010): http://www.netlib.org/hompack/. Accessed November 2010.

Horowitz P., Hill W. (1989): **The Art of Electronics**. Cambridge University Press, page 189.

Keller H. (1978): **Global homotopies and Newton methods**. C. de Boor and G. Golub, eds., Academic Press, New York, pp. 73-94.

Kelley C.T. (2003): **Solving nonlinear equations with Newton's method**. SIAM.

Mattsson S.E., Elmqvist H., Otter M., Olsson H. (2002): **Initialization of Hybrid Differential-Algebraic Equations in Modelica 2.0**. Proceedings of the Second International Modelica Conference, Munich, Germany, pp. 9-15. Download: https://www.modelica.org/events/Conference2002/papers/p02_Mattsson.pdf

Modelica (2010): **Modelica – A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2**. March 24. Download: https://www.modelica.org/documents/ModelicaSpec32.pdf

Tietze U., Schenk C. (2002): **Halbleiterschaltungstechnik**. Springer, 12th edition, page 1150.

Watson L.T., Billups S.C., Morgan A. P. (1987): **Algorithm 652: HOMPACK, A suite of codes for globally convergent homotopy algorithms**. ACM Transactions on Mathematical Software, 13, 281-310.

Wayburn T., Seader J. (1987): **Homotopy continuation methods for computer-aided process design**. Computers & Chemical Engineering 11, pp. 7-25.