Robust solution of inverse static problems in Modelica: the case for the initialSimplified() operator

Francesco Casella[1] Philip Hannebohm[2] Matteo Luigi De Pascali[1]

[1]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy,
{francesco.casella,matteoluigi.depascali}@polimi.it
[2]Institute for Data Science Solutions, Bielefeld University of Applied Sciences and Arts, Germany,
philip.hannebohm@hsbi.de

Abstract

Solving inverse static initialization problems, in which the outputs of a process are prescribed, is often more numerically challenging than solving direct initialization problem, where inputs are fixed instead. This paper proposes the introduction of a new `initialSimplified()` operator in Modelica, which allows to solve inverse problems with the same reliability as direct problems, using an intuitive, straightforward Modelica formulation. The proposed solution can be trivially implemented in Modelica tools and will be 100% portable among them, once the new operator is standardized. The usefulness of the proposed approach is demonstrated on a conceptual thermal power generation plant model.

Keywords: Modelica language, initialization, homotopy, convergence problems

1 Introduction

The Modelica language and tools are widely used for the dynamic modelling of industrial processes and energy conversion systems, which often need to be initialized in steady-state. They are also increasingly considered as a convenient framework to formulate and solve static models of such processes, see, e.g., (Luiten et al. 2026), where they can be very convenient because most of the model equations can be used for process design, where certain process parameters are unknown and need to be determined, and then for off-design static analysis of the process behaviour, where the parameters are fixed and some different boundary conditions can be provided.

One major issue in both cases is that large, strongly non-linear systems of algebraic equations stemming from those problems need to be solved with iterative methods, which may fail if the supplied initial guess is not close enough to the solution. Without proper ways to address this issue systematically, the potential for practical application of Modelica in this context is severely hindered.

Homotopy-based methods have been proposed and successfully used to mitigate the problem (Sielemann, Casella, Otter, et al. 2011; Casella et al. 2011; De Pascali and Casella 2026). The basic idea is to initially replace some expression $f(x)$ with a simplified counterpart $g(x) = 0$, so that the corresponding simplified problem can be solved in a more robust and reliable way than the original problem, then gradually morph the simplified expression into the actual one by means of the homotopy transformation

$$f_h(x) = \lambda f(x) + (1-\lambda)g(x) \tag{1}$$

where λ goes from 0 to 1. As discussed in (De Pascali and Casella 2026), the solution of the problem at $\lambda = 0$ can be made more robust by means of two strategies:

- replacing strongly non-linear expressions $f(x)$ with mildly non-linear or linear simplified versions;
- replacing time-varying terms in $f(x)$ with constant, known terms, e.g. some nominal or design value.

The first strategy aims at widening the set of initial guesses from which iterative solvers can successfully converge to the solution, possibly allowing to use default values that do not require specific tuning by the end-user, or even removing the need for initial guess values entirely, in case one obtains a linear system at $\lambda = 0$. The second strategy, besides also allowing to make the problem less non-linear, also allows to split the original, fully coupled problem into a sequence of smaller problems that can be solved sequentially. As clearly demonstrated in (De Pascali and Casella 2026), both strategies can lead to demonstrably huge benefits in terms of convergence robustness.

Note that, in order to fully benefit from the robustness improvements brought by these strategies, it is essential to perform ad-hoc structural analysis and symbolic simplification of the initialization problem at $\lambda = 0$, generating specific code to solve the simplified problem. This fact has been widely recognized for a long time and popular Modelica tools such as Dymola or OpenModelica implement it.

Of course, one must also choose the simplified expression $g(x)$ wisely, so that the solution of the problem as λ goes from 0 to 1 is devoid of singularities and as smooth as possible; however, this aspect goes beyond the scope of this paper.

In fact, there is one interesting use case, which is discussed in (De Pascali and Casella 2026), for which the

Modelica language currently does not provide proper support. Process models are characterized by input and output variables: inputs can be set by means of actuators, and outputs can be measured, and possibly controlled to specific target values by means of control systems. When solving steady-state initialization problems for dynamic models, or off-design problems for static process models, it is very often the case that the *forward* initialization problem, where the inputs are fixed, leads to a simplified model at $\lambda = 0$ which is much easier to solve than the *backward* problem, where one fixes the outputs of the system, in order to compute the corresponding inputs as a result. As a consequence, the solution of *backward* problems tends to be less numerically robust and more prone to convergence failure at $\lambda = 0$ than the solution of *forward* problems.

For example, if the input variables of a thermo-fluid process are certain fluid flow rates, fixing them makes the problem much less non-linear, because many quantities such as heat transfer coefficients or friction losses can be computed explicitly from those flow rates; furthermore, originally non-linear equations such as energy balances, which contain the product of flow rate times specific enthalpy, become linear once the flow rate is known. If some output variables like outlet temperature or turbine output power are fixed instead, the flow rates will be unknown, so the non-linear system to be solved will be larger and more non-linear.

An effective strategy to avoid the reduction of convergence robustness when solving *backward* problems is to solve the initialization in three steps:

1. Solve the on-design *forward* problem at $\lambda = 0$, by fixing the design values of the inputs.

2. Solve the on-design *backward* problem at $\lambda = 0$,[1] by fixing the design values of the outputs, using the previously found solution as an initial guess. Since the two solutions are identical, or very close to each other, no convergence problems arise.

3. Use the homotopy transformation to gradually move the solution from the on-design output values to the desired off-design values.

By following this strategy, the successful convergence of the more robust *forward on-design* initialization problems is enough to ensure the convergence of *backward on-design* initialization problems, thus eliminating the previously mentioned issue.

Unfortunately, it is currently not possible to program such a solution strategy directly in Modelica; one needs to use some scripting environment and Modelica tool API calls to first perform step 1, save the simulation results, and then reload them as an initial guess for step 2. This approach is not very convenient and, most importantly, not portable between different Modelica tools, since scripting languages and APIs are not standardized.

The goal of this paper is thus to motivate the introduction of a new language feature in Modelica, namely the operator `initialSimplified()`, which makes it possible to perform the three above-mentioned steps directly in any Modelica tool, with no need for scripting.

The paper is structured as follows: Section 2 introduces the new operator, its semantics, and how it can be implemented in Modelica tools; Section 3 describes a test case that validates the usefulness of the proposed approach, while Section 4 presents experimental results obtained with that test case. Section 5 concludes the paper with final considerations and recommendations for future work.

2 Method

2.1 Definition of the new operator

A new built-in Modelica operator with function syntax `initialSimplified()` is introduced.

This operator returns `true` when the simplified $\lambda = 0$ initialization problem is being solved, `false` otherwise. The intended usage of this operator is as a condition for if-equations and if-expressions, e.g.

```
if initialSimplified() then
  // Equations to be considered when
  // generating the code to solve the
  // simplified \lambda = 0 problem
else
  // Equations to be considered in all
  // other cases
end if;
```

The Modelica tool will consider the first set of equations to generate the code for the simplified $\lambda = 0$ initialization problem, while it will consider the second set of equations to generate the code that performs the homotopy transformation during initialization, as well as to generate the code for simulation.

The behaviour of this operator is inspired from and similar to the existing `initial()` operator. If one writes

```
if initial() then
  // Equations to be considered during
  // initialization
else
  // Equations to be considered in all
  // other cases
end if;
```

a Modelica tool will generate the code to solve the initialization problem with the first set of equations, computing the values of parameters and the initial values of states. Then, when the simulation starts, it will solve the simulation problem using the other set of equations at `time = StartTime`, fixing the state and parameter values and using the previously computed values of unknown algebraic variables as initial guesses for iterative solvers. The use case is also similar: the first set can be formulated to

[1] At least conceptually, in practice one may skip that solve and directly continue with the first $\lambda > 0$.

DOI
10.3384/ecp22331

make the initialization problem easier to solve when coupled to the initial equations, but still with a formulation that is equivalent (or nearly equivalent) to the second set, so that the first simulation step will compute the same (or very close) values starting from the previous solution as initial guess, thus with no convergence issues in case iterative solvers are involved.

2.2 Mathematical formulation

The first step of the simulation process for any Modelica model is the solution of the initialization problem. Considering for simplicity a purely continuous-time model with no discrete and clocked variables ad no algorithm sections, the unknowns of the initialization problem are the initial values of the state variables x, of their derivatives $\dot{x}$, of the algebraic variables v, and of the unknown parameters p with `fixed = false` attribute. The corresponding set of equations is obtained by assembling the **`equation`** and `initial` **`equation`** sections of the flattened model:

$$F(x,\dot{x},v,p,t,\lambda) = 0, \tag{2}$$

which is then solved at time $t =$ startTime. If the `homotopy()` operator is used in the model formulation, the equations in the initialization problem (2) depend on the homotopy parameter λ as recalled in Eq. (1).

As extensively discussed in (Sielemann, Casella, Otter, et al. 2011; Casella et al. 2011; Sielemann, Casella, and Otter 2013), the problem (2) is first solved for $\lambda = 0$, exploiting all structural simplifications that take place for that value of λ, using the `start` attributes of the unknowns as initial guesses; then, λ is continuously transformed from 0 to 1 (possibly in a non-monotonous way) to eventually achieve the solution of the actual problem at $\lambda = 1$. The idea is to formulate the problem at $\lambda = 0$ so that its convergence is less sensitive (or not sensitive at all, ideally) to a proper selection of initial guesses than the solution of the actual initialization problem. However, this may not always be possible for structural reasons, e.g. when solving backward initialization problems.

With the introduction of the `initialSimplified()` operator, the equations of the initialization problem can be split into three subsets:

$$F_r(x,\dot{x},v,p,t,\lambda) = 0 \tag{3}$$
$$F_s(x,\dot{x},v,p,t) = 0 \tag{4}$$
$$F_n(x,\dot{x},v,p,t,\lambda) = 0, \tag{5}$$

where (3) includes all equations that do not contain `initialSimplified()`, (4) includes the equations containing `initialSimplified()`, assuming it returns `true` and (5) includes the same equations, assuming that `initialSimplified()` now returns `false`.

The solution of the initialization problem is now split into the sequential solution of three problems:

1. Solve Eq's (3), (4) for $\lambda = 0$, $t =$ startTime, using the `start` attributes of the unknowns x, v, and p as initial guesses for the iterative solvers.
2. Solve Eq's (3), (5) for $\lambda = 0$, $t =$ startTime, using the solution found at the previous step as an initial guess for the iterative solvers.
3. Perform the homotopy transformation of Eq's (3), (5) for λ going from 0 to 1 as usual.

For both the first and second problems, it is essential to exploit all the structural simplifications made possible by the specific problem formulation.

The main idea is to use the `initialSimplified()` operator to formulate the first problem in a way can be solved more robustly than the second (which is what you currently get with vanilla `homotopy`), while leading to the same solution, or to a very near solution, so that the convergence of the solver on the second step will be guaranteed.

In particular, this approach can be readily applied to the steady-state initialization of dynamic process models: the first problem is the forward on-design initialization, where the design values of the *inputs* are enforced, while the second problem is the backward on-design initialization, where the design values of the *outputs* are applied, ideally leading to the same solution, or to a very close one. The third step can then be used to gradually drive the solution of the backward initialization problem from on-design to off-design conditions.

2.3 Implementation in Modelica tools

The implementation of the `initialSimplified()` operator is trivial. As noted in the Introduction, Modelica tools already split the code generation for initialization between the $\lambda = 0$ and the $\lambda > 0$ cases, so they just need to replace the new operator by `true` when generating the code for $\lambda = 0$, by `false` otherwise. That's it.

Note that the two branches of if-equations or if-expressions depending on this operator can lead to systems that have a completely different structure, e.g. *forward* vs. *backward* initialization. This is indeed intended; in fact, it is the whole point of the proposal. On the other hand, for this approach to work, it is essential that the two systems have the same (or nearly the same) solution, to ensure that the solution found at the first step can be used as an effective initial guess to solve the second; obviously, this is entirely the responsibility of the modeller.

2.4 User-friendly application in Modelica models

The open-source BoundaryInitBlocks Modelica library (De Pascali and Casella 2024), presented in (De Pascali and Casella 2026), provides general-purpose blocks to manage steady-state initialization of process models, allowing the end user to easily set up forward and backward initialization problems, by setting suitable parameters in blocks connected to the process input and output connectors.

In that paper, the possibility of supporting robust backward initialization by first solving the forward

on-design problem is discussed; amending the library code to provide such a feature with the proposed `initialSimplified()` would be trivial.

3 Test Cases

The usefulness of the proposed `initialSimplified()` operator is demonstrated by means of the conceptual thermal power generation system model listed in Fig. 1.

The model represents a counter-current heat exchanger where a hot flue gas with temperature T_{fg} heats a gas working fluid with temperature T, which is then expanded in a turbine to produce a power output P. The heat exchanger is modelled by N finite volumes.

The model is fully static, so it is representative of both static design models and steady-state initialization of a corresponding dynamic model, where all the derivative terms are set to zero by the initial equations.

The two inputs of the process are the mass flow rate w of the working fluid, which is supplied by a compressor, and the mass flow rate w_{fg} of the flue gas, which is supplied by the exhaust of some combustion process.

The forward initialization problem can be formulated by fixing the two inputs to their design values:

```
model ForwardOnDesign
  Plant plant;
equation
  plant.w = 1;
  plant.wfg = 1;
end ForwardOnDesign;
```

Since w and w_{fg} are set to known values, one can explicitly compute vector G; then, the first three equations of the for-loop form a system of $3N$ equations that can be solved by a linear solver, so there is no need to set any guess value for the unknown vectors T, T_{fg}, Q. Once that is done, all the remaining equations can be solved explicitly.

Hence, despite the obvious non-linearity of the process model, as long as the inputs are fixed it is possible to solve it in a completely reliable way, without the need of worrying about initial guess values.

Consider now the following *backward* initialization problem, where we instead fix two outputs, the flue gas outlet temperature T_s and the turbine power P. Note that the second output is fixed to an off-design value which is 30% of the nominal design value; however, homotopy is used to get there gradually, starting in nominal conditions at $\lambda = 0$; this is done as suggested in (De Pascali and Casella 2026) because the solution of on-design problems is typically less numerically challenging than in strongly off-design conditions.

```
model BackwardOffDesign
  Plant plant;
  parameter Real load = 0.3;
equation
  plant.P = homotopy(
    actual = plant.Pnom*load,
    simplified = plant.Pnom);
  plant.Ts = plant.Tsnom;
end BackwardOffDesign;
```

In this case, both fluid mass flow rates w and w_{fg} are unknown, so it is not possible to compute the conductances G a-priori. Due to the structure of the problem, one now gets a slightly larger implicit system of $3N+3$ equations that also includes the two turbine equations and one scalar equation for G (the other $N-1$ can be removed by common sub-expression elimination). Contrary to the previous *forward* problem, this system is now strongly non-linear. Solving it without setting initial guesses for all these unknowns is very likely going to be problematic.

With the proposed new `initialSimplified()` operator, one can write the following model:

```
model BackwardOffDesignInitialSimplified
  Plant plant;
  parameter Real load = 0.3;
equation
  if initialSimplified() then
    plant.w = 1;
    plant.wfg = 1;
  else
    plant.P = homotopy(
      actual = plant.Pnom*load,
      simplified = plant.Pnom);
     plant.Ts = plant.Tsnom;
  end if;
end BackwardOffDesignInitialSimplified;
```

According to the proposed semantics, the code to solve the problem at $\lambda = 0$ is generated with fixed inputs, so it will have exactly the same structure as in `ForwardOnDesign`; hence, it will be solved reliable without the need of setting initial guess values. The initialization problem will then be solved with a homotopy transformation that involves the non-linear system of $3N+3$ equations; however, at $\lambda = 0$ that system still has a solution that is very close to the solution of the *forward* simplified initialization problem that has just been solved, so if that solution is used to set initial guess values, the iterative Newton solver will converge easily. Then, the solution will be gradually shifted to 30% partial load, but this will not be problematic as long as short enough λ-steps are taken.

This simple example clearly shows the advantages of the proposed approach:

- The Modelica code is very straightforward and intuitive to write. As noted in Section 2.3, this code could written once and for all in reusable utility blocks.
- The much harder non-linear *backward* initialization problem will be solved with the same robustness as the *forward* problem, without the need of setting any additional start attributes.
- Once the new operator is standardized, this model will be portable across Modelica tools, with the same guaranteed degree of robustness.

The authors tried hard to test this approach in full-scale, realistic test cases, using the fuel cell and innovative power generation plant models that are discussed in (De Pascali and Casella 2026). However, implementation issues

DOI
10.3384/ecp22331

with the new backend of the OpenModelica compiler at the time of this writing prevented the successful testing in those cases; this will be the subject of a subsequent publication.

4 Experimental results

Experiments were carried out with the test case illustrated in the previous Section, using Dymola 2026x and OpenModelica v1.27.0-dev-462, which implements the experimental `initialSimplified()` operator in the new backend.

As expected, due to its nice structural properties, the `ForwardOnDesign` model is solved successfully by both tools.

When tackling the `BackwardOffDesign` model, Dymola fails to solve it at $\lambda = 0$, because of repeated division-by-zero errors, due to the lack of reliable initial guess values for the solution of the implicit *backward* initialization problem. OpenModelica barely manages to solve it, after repeatedly reporting numerical issues at $\lambda = 0$ such as division by zero or raising a negative number to a Real power. As expected, the solution of the *backward* initialization problem at $\lambda = 0$, which involves a significantly non-linear system of equations, can be problematic if initial guess values are not carefully provided.

The `BackwardOffDesignInitialSimplified` model is instead solved successfully by the experimental implementation in OpenModelica without any warnings indicating potential numerical issues, as expected from the analysis carried out in the previous Section.

This confirms the main argument of this paper, namely that the availability of the `initialSimplified()` operator allows to solve *backward* initialization problems with the same reliability of *forward* problems, in a way that would be 100% portable across tools, once the operator is standardized.

Regarding tests on full-scale power generation models, the currently available experimental evidence is that OpenModelica solves the forward initialization problems for those plant models easily, while it struggles with the backward ones; hence, it is expected that the application of the approach proposed in this paper to manage the initialization would be beneficial. However, it was not possible to directly test the proposed approach at the time of this writing due to (unrelated) issues with the new OpenModelica backend, where the experimental operator is currently implemented.

5 Conclusions

The solution of *backward* initialization problems in Modelica process models is often more numerically challenging than the solution of *forward* initialization problems. An effective strategy to avoid this problem is to use homotopy-based initialization, solving the *forward* problem in on-design conditions at $\lambda = 0$ first, then using the result as an initial guess for the homotopy transformation of the full *backward* initialization problem, which starts from the same on-design conditions (so convergence is guaranteed) and gradually shifts to off-design conditions.

This paper proposes the introduction of the operator `initialSimplified()`, which allows to formulate such a strategy directly in the Modelica model in a very intuitive way, without the need of performing complex operations with scripting languages and tool APIs, that are not portable across Modelica tools. Such a strategy can be very effective at improving the robustness of convergence of *backward* problems, avoiding or drastically reducing the need of setting start attributes to provide good initial guesses to the iteration variables of the solver, which is a tedious and fragile process, also due to the fact that different Modelica tools (or even different versions of the same tool) may choose different iteration variables for the same problem.

The implementation of the `initialSimplified()` operator is trivial for tools that already generate separate code for the initialization problem at $\lambda = 0$, in order to exploit all the structural simplifications introduced by the homotopy simplifications.

The effectiveness of this approach was demonstrated successfully on a conceptual model of a thermal power generation system. At the time of this writing, it was not yet possible to test this approach directly on full-scale power generation system models because of (unrelated) implementation issues with the new OpenModelica backend, where the experimental operator is implemented. However, the experimental evidence with the old backend shows that solving forward problems is much less problematic than solving backward problems, thus suggesting that the proposed approach will be beneficial.

The authors thus recommend the inclusion of the new operator in the forthcoming release of the Modelica Language Specification.

Acknowledgements

The authors would like to express their sincere appreciation to the Open Source Modelica Consortium (OSMC) for support, collaboration, and shared commitment to advancing open-source modelling and simulation technologies.

References

Sielemann, Michael, Francesco Casella, and Martin Otter (2013). “Robustness of declarative modeling languages: Improvements via probability-one homotopy”. In: *Simulation Modelling Practice and Theory* 38, pp. 38–57. DOI: 10.1016/j.simpat.2013.07.001.

Sielemann, Michael, Francesco Casella, Martin Otter, Christoph Clauß, Jonas Eborn, Sven Erik Mattsson, and Hans Olsson (2011-03). “Robust Initialization of Differential-Algebraic Equations Using Homotopy”. In: *Proceedings 8th International Modelica Conference*. Ed. by C. Clauss. Modelica Association. Dresden, Germany, pp. 75–85. ISBN: 978-91-7393-

096-3. DOI: 10.3384/ecp1106375. URL: http://www.ep.liu.se/ecp/063/010/ecp11063010.pdf.

Casella, Francesco, Michael Sielemann, and Luca Savoldelli (2011-03). "Steady-state initialization of object-oriented thermo-fluid models by homotopy methods". In: *Proceedings 8th International Modelica Conference*. Ed. by C. Clauss. Modelica Association. Dresden, Germany, pp. 86–96. ISBN: 978-91-7393-096-3. DOI: 10.3384/ecp1106386. URL: http://www.ep.liu.se/ecp/063/011/ecp11063011.pdf.

De Pascali, Matteo Luigi and Francesco Casella (2024). *BoundaryInitBlocks library*. https://github.com/looms-polimi/BoundaryInitBlocks.

De Pascali, Matteo Luigi and Francesco Casella (2026). "Robust steady-state initialization of object-oriented advanced thermal power generation system models with application to the case of the SOS-CO_2 cycle". In: *Mathematical and Computer Modelling of Dynamical Systems* 32.1, p. 2616725. DOI: 10.1080/13873954.2026.2616725.

Luiten, Casper, Carlo De Servi, and Francesco Casella (2026). *Combined On-Design and Off-Design Modeling of Aerospace- grade Thermal Systems in Modelica*. https://openmodelica.org/images/M_images/OpenModelicaWorkshop_2026/OpenModelica2026_Casper_Luiten.pdf. Presentation at the 2026 OpenModelica Workshop.

Figure 1. Conceptual thermal power generation model

```
model Plant
  connector RealInput = input Real;
  connector RealOutput = output Real;

  parameter Integer N = 5 "Number of finite volumes";
  parameter Real NTU = 4 "Nominal NTU";
  final parameter Real Gtot = NTU "Nominal total heat conductance";
  parameter Real cp = 1 "Specific heat capacity";
  parameter Real kf = 0.1 "Fluid friction coefficient";
  parameter Real Kt = 1 "Turbine flow coefficient";
  parameter Real Tnom = 700 "Nominal TIT";
  parameter Real Tsnom = 400 "Nominal flue gas outlet temperature";
  parameter Real Pnom = 223 "Nominal turbine power";
  parameter Real pnom = 1.1 "Nominal fluid inlet pressure";
  parameter Real TOP = 0.1 "Nominal turbine outlet pressure";
  parameter Real gamma = 1.2 "cp/cv";

  RealInput w "Fluid flow rate";
  RealInput wfg "Flue gas flow rate";

  Real T[N + 1] "Fluid temperatures";
  Real p[N + 1] "Fluid pressures";
  Real Tfg[N + 1] "Flue gas temperatures";
  Real G[N] "Volume conductivity";
  Real Q[N] "Thermal powers";
  Real TIT = T[N + 1] "Turbine inlet temperature";

  RealOutput P "Turbine power";
  RealOutput Ts = Tfg[N+1] "Outlet flue gas temperature";
  RealOutput TIP = p[N + 1] "Turbine inlet pressure";
  RealOutput pin = p[1] "Fluid inlet pressure";
equation
  T[1] = 300;
  Tfg[1] = 800;
  for i in 1:N loop
    0 = w*cp*(T[i] - T[i + 1]) + Q[i];
    0 = wfg*cp*(Tfg[N - i + 1] - Tfg[N - i + 2]) - Q[i];
    Q[i] = ((Tfg[N - i + 2] + Tfg[N - i + 1])/2 - (T[i + 1] + T[i])/2)*G[i];
    G[i] = NTU/N*w^0.8;
    (p[i] - p[i + 1]) = kf/N*w^2;
  end for;
  w = Kt*p[N + 1]/sqrt(TIT/Tnom);
  P = w*cp*TIT*(1 - (TOP/TIP)^((gamma - 1)/gamma));
end Plant;
```

DOI
10.3384/ecp22331