An Environment for Testing, Verification and Validation of Dynamical Models in the Context of Solid Oxide Fuel Cells*

Stefan Kiel, Ekaterina Auer
Computer and Cognitive Sciences (INKO),
University of Duisburg-Essen, Duisburg, Germany
(kiel,auer)@inf.uni-due.de

Andreas Rauh
Chair of Mechatronics,
University of Rostock, Rostock, Germany
andreas.rauh@uni-rostock.de

Abstract

Still in its early development stages, the technology of solid oxide fuel cells is one of the important topics in modern engineering. One research direction is to design robust and accurate control strategies for this kind of fuel cell based on models spatially discretized into ordinary differential equations that describe also non-stationary system behavior. To allow users to employ new models and techniques for SOFCs easily in combination with various verified tools, we implement the environment VeriCell. It features an intuitive graphical interface for construction of fuel cell models from predefined building blocks. It is based on the framework UniVerMeC which provides a unified access to various verified arithmetics and algorithms. New models can be added to VeriCell as they are being developed, for which purpose a plug-in based interface is adopted. In this paper, we present the environment, with the focus on the software design and verified simulation of initial value problems. The main features of the software are demonstrated on examples modeled and simulated in VeriCell.

Keywords: solid oxide fuel cells, initial value problems, software engineering, verification, validation, UniVerMeC

AMS subject classifications: 65-06, 65G20, 65P99, 65Z99

*Submitted: February 10, 2013; Revised: May 6, 2014; Accepted: May 6, 2014.

1 Introduction

Solid oxide fuel cells (SOFCs) are devices that convert chemical energy into electricity. They are relatively compact, highly efficient, and flexible with respect to the kind of fuel. A drawback is the complex production process. Moreover, SOFCs might suffer from overheating, which needs to be avoided. These reasons have stimulated a lot of research in the area [4,17], in which scientists typically use partial differential equations based models to describe the behavior of SOFCs [22]. However, the technology is still in its early development stages.

One approach to deal with the problems is to design and develop robust and accurate control strategies for SOFCs which are based on mathematically more simple ordinary differential equation (ODE) models. This is a major goal of a current joint project between the Universities of Rostock and Duisburg-Essen. State-of-the-art SOFC control methods do not take non-stationary operating points into account and are thus valid only in small operation ranges. Recently, we have presented control oriented SOFC temperature models and new control strategies for them [6, 5, 20]. The simulations are valid in wide operation ranges, and make partial use of methods from verified numerics (in particular, interval arithmetic) to provide *reliable* control strategies taking into account bounded uncertainty and disturbance. Reliability is especially important in the SOFC area since it is necessary to guarantee that overheating does not happen and so cannot damage the expensive cell.

Developing and testing control strategies with different models is a complex process. The first step is obviously to choose the right kind of model, in our case, a control oriented one. Usually, it is then necessary to adapt the selected model to the considered *real* SOFC. This can be done by parameter identification, which is applied to the model on the basis of measured data from the cell obtained over a long period of time. Experiment design is also of interest, and can be seen as an optimization problem. In the final step, users might want to validate the chosen model and parametrization by checking the resemblance between the simulation and measured results or by assessing the performance of the employed control strategy. One possibility to validate in such a way is to actually run a series of test simulations, which amounts to solving initial value problems (IVPs) in our context.

To allow users to employ new models and techniques easily in combination with various verified tools, we implement the environment VeriCell within the scope of the joint research project `VeriIPC-SOFC`. Figure 1 gives an overview of the project and highlights the parts relevant to this paper. The already mentioned project tasks are development of SOFC models and control strategies for them as well as finding means for accurate identification of their parameters, for simulation and for sensitivity analysis. For this purpose, we (plan to) employ a variety of methods from the areas of traditional and verified numerics (possibly, in combination with each other). Here, we need at least methods for solving algebraic, differential, and/or differential-algebraic equations as well as optimization algorithms. Stochastic approaches can be used to model processes such as the aging of an SOFC stack. All these tasks are used for and can be validated against the SOFC test rig available at the University of Rostock. Finally, a certain expertise in the area of software engineering is necessary to be able to develop an integrated simulation environment interacting with the test rig, since common interfaces between different approaches are lacking (in particular, for verified implementations).

The VeriCell software supports users during the task of testing new SOFC control strategies and models. It features an intuitive graphical interface for construction

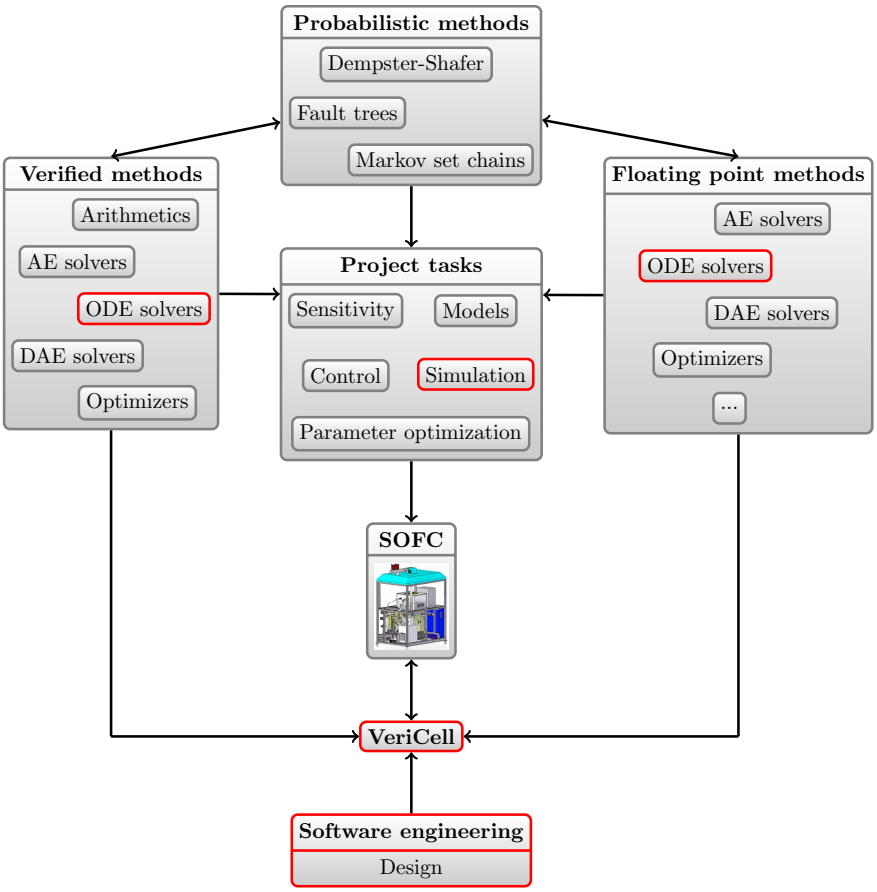


Figure 1: Overview of the `VerIPC-SOFC` project. The parts discussed in this paper are shown in red.

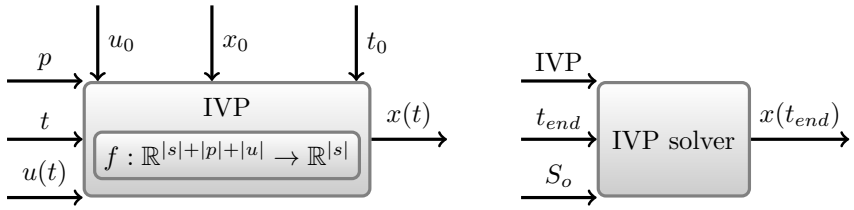
of SOFC models from predefined building blocks. New models can be added to VeriCell as they are being developed. Moreover, users can apply a GPU-accelerated parameter identification algorithm [12] with their own measured data to adapt the models to their actual SOFCs. To simulate controlled or non-controlled models, several external initial value problem solvers can be employed inside VeriCell. Among these are verified (ValEncIA-IVP [18] and VNODE-LP [16]) as well as non-verified ones (currently, ODE and VODE [1]). An important design aspect of VeriCell is flexibility with respect to external solvers and basic data types. The users are able to incorporate new models and interface new solvers through a mechanism based on the framework UniVerMeC [8]. It not only provides a unified access to various arithmetics and algorithms but also centralizes the model description inside the environment. That is, the same model definition (e.g. an IVP defined according to the VeriCell specifications) can be used with all the supported solvers.

Verification in the context of SOFC modeling, simulation, and control does not primarily mean a guarantee of a certain number of digits after the decimal point. Since we apply methods of verified numerics to a *model* of a real-life SOFC stack, we can only certify that the exact solution to *this model* is within the bounds obtained on a computer. By using the term 'verification', we emphasize, on the one hand, that the rounding and numerical discretization errors (within the restrictions of the considered model) are taken into account. In this way, we are sure that no uncertainty is added to the model due to numerics, even if parameters of the model are not known exactly. On the other hand, this is a term which denotes the set-based methods with result verification in general, for example, interval, affine or Taylor model analysis. These set-based methods help us to propagate bounded uncertainty through the system without adding further numerical uncertainty, as explained above.

The paper is structured as follows. First, we briefly describe SOFC models currently available in VeriCell. Next, we summarize the main software engineering principles of UniVerMeC in Section 3. In the next section, we give details on how to define a model in and to interface a solver using UniverMeC. In Section 5, the main features of VeriCell are demonstrated on several simulation examples. A discussion of the results and a perspective on future work conclude this paper.

2 Models

An SOFC model adopted in [6] consists of three main parts, each describing a specific aspect of the system behavior: its fluid dynamics, electrochemistry and thermodynamics. At the moment, our research is concentrated on thermal models, since this is the most influential component. The phenomenon can be described by nonlinear ODEs obtained using the finite volume method by spatial semi-discretization of the stack into $L \times M \times N$ volume elements. For example, the $1 \times 1 \times 1$ model [19] characterizes



(a) Abstract IVP class used for representing model components.

(b) Representation of an IVP solver and the necessary input parameters.

Figure 2: The two main components necessary for simulating SOFCs. The abstract description of a component acts as an input for an IVP solver.

the cell temperature θ_{FC} by

$$\begin{aligned}
\dot{\theta}_{FC} &= \dot{m}_{H_2} \cdot \left(p_{\Delta H,2} \cdot \theta_{FC}^2 + p_{\Delta H,1} \cdot \theta_{FC} + p_{\Delta H,0}\right) + 6 \cdot p_A \cdot (\theta_A - \theta_{FC}) \\
&+ (\theta_{AG} - \theta_{FC}) \cdot \left(\dot{m}_{H_2} \cdot (p_{H_2,2} \cdot \theta_{FC}^2 + p_{H_2,1} \cdot \theta_{FC} + p_{H_2,0})\right. \\
&+ \dot{m}_{H_2O} \cdot (p_{H_2O,2} \cdot \theta_{FC}^2 + p_{H_2O,1} \cdot \theta_{FC} + p_{H_2O,0}) \\
&+ \left.\dot{m}_{N_2} \cdot (p_{N_2,A,2} \cdot \theta_{FC}^2 + p_{N_2,A,1} \cdot \theta_{FC} + p_{N_2,A,0})\right) + I_{FC} \cdot p_{el} \\
&- \dot{m}_A \cdot (\theta_{FC} - \theta_{CG}) \cdot \left(77 \cdot p_{N_2,C,0}/100 + 11 \cdot p_{O_2,0}/50 + 77 \cdot p_{N_2,C,1} \cdot \theta_{FC}/100\right. \\
&+ \left.11 \cdot p_{O_2,1} \cdot \theta_{FC}/50 + 77 \cdot p_{N_2,C,2} \cdot \theta_{FC}^2/100 + 11 \cdot p_{O_2,2} \cdot \theta_{FC}^2/50\right)
\end{aligned} \tag{1}$$

with the initial condition $\theta_{FC} = 299.7053$K. Parameters denoted by $\dot{m}$ describe mass flows of, for example, nitrogen and are assumed to be piecewise constant. The parameters p are constant and represent approximations to the temperature-dependent heat capacities of hydrogen, nitrogen, water vapor and air as well as the reaction enthalpy and material and thermal resistances of the cell [3]. We omit further details on the model parameters such as their actual values since they are not relevant for our present discussion.

On an abstract level, an IVP we have to deal with in VeriCell is composed of the components shown in Figure 2a. The right-hand side of the ODEs has to be specified. It can be seen as a function f depending on a number of constant parameters p and piecewise constant, time-varying external signals $u(t)$. This leads to the model equation shown below

$$\dot{x} = \underbrace{f(x, p, u(t))}_{\mathbb{R}^{|s|+|p|+|u|} \to \mathbb{R}^{|s|}}, \tag{2}$$

where $|s|$ is the dimension of the state vector x, and $|p|$ and $|u|$ are the dimensions of the constant and time-dependent parameter vectors. (Note that the $u(t)$ are not treated simply as a function of time, being a part of the right-hand side f, but as parameters constant during each time step and possibly changing in between.) Together with initial values t_0, u_0, and x_0, the equation (2) forms the basis for the abstract class representing the IVP in VeriCell shown in Fig. 2a. If we want to interface an external IVP solver, it is sufficient to implement a mapping from the abstract IVP representation to the solver's internal one. As shown in Fig. 2b, the class for introducing an IVP solver into VeriCell takes as its input the maximum integration time t_{end} and solver specific parameters S_O in addition to the abstract IVP representation.

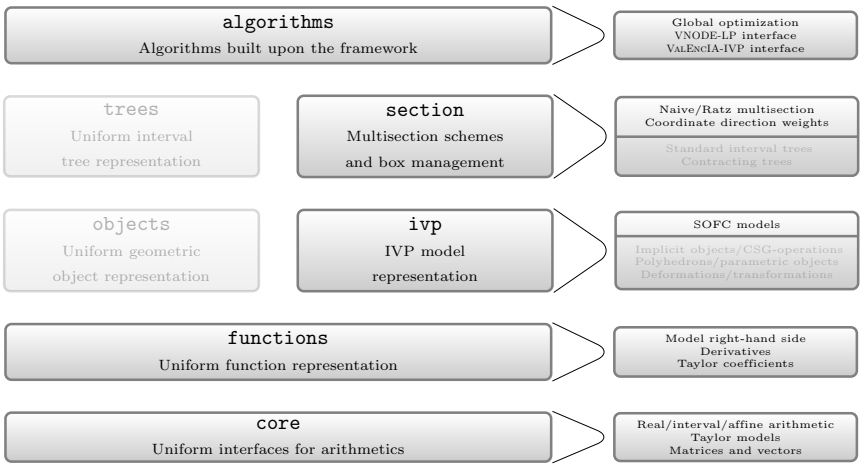


Figure 3: Relaxed layered structure of the UniVerMeC framework and its application in the scope of VeriCell.

Currently, VeriCell supports ValEncIA-IVP and VNODE-LP as external verified IVP solvers. Furthermore, non-verified simulation results can be obtained by ODE, VODE or the use of a "verified" approximation of the exact solution by Euler's method (needed for parameter identification)

$$\boldsymbol{x}_k := \boldsymbol{x}_{k-1} + h \cdot f(\boldsymbol{x}_{k-1}, p, u(t))$$

which does not account for the local discretization errors but deals with the rounding errors and parameter uncertainties. Although the simplest interval version of the Euler method suffers considerably from the wrapping effect, we actually have no choice but to use it for parameter identification in the set-based case since every other possibility to express the solution to the IVP (apart from the analytical expression which is not available) takes too much CPU time [2]. This is also one of the reasons why the high dimensional models do not correspond with the reality as well as the one dimensional model (cf. Section 5).

Depending on the chosen solver, it is necessary to access different kinds of information stored in the abstract IVP representation. For example, if we use an interval IVP solver, we need to be able to evaluate the right-hand side f with interval arithmetic, whereas solvers like VODE evaluate it using standard floating point arithmetic. Another example for the differences in the required information is that ValEncIA-IVP uses only the Jacobian of f whereas VNODE-LP needs Taylor coefficients of both f and its Jacobian. That is, the major task for us to deal with consists of representing f in such a way to allow for ready availability of such information. Moreover, it should be possible to extend the representation if additional features are required.

3 UniVerMeC

To solve the aforementioned task of representing the right-hand side f and allowing for computations with different arithmetics, we employ our framework UniVerMeC [8,13]

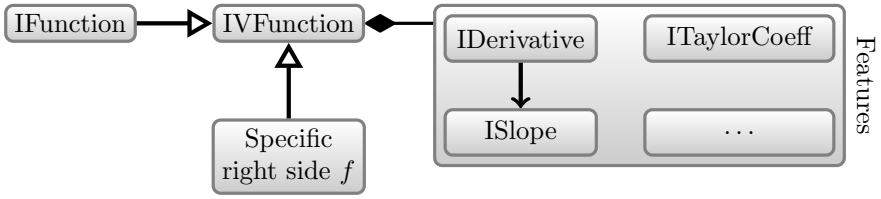


Figure 4: Representation of functions in UniVerMeC.

in VeriCell. It was developed within the scope of verified geometric computations, and uses the relaxed[1] layered structure shown in Figure 3. The bottom layer `core` provides access to floating point, interval, and affine arithmetic as well as to Taylor models. All arithmetic types share a common interface. At the `function` layer, a uniform representation of functions in their mathematical sense is provided both for scalar valued $f_s : \mathbb{R}^n \to \mathbb{R}$ and vector valued $f_v : \mathbb{R}^p \to \mathbb{R}^q$ ones. This concept allows us to evaluate a function so represented with all arithmetics supported at the `core` layer. Further, abstractions for derivatives, slopes, Taylor coefficients or contractors are provided at this level, a list which can be extended if necessary.

The third layer is responsible for defining models in the framework, for example, geometrical objects or IVPs. It merges the relevant abstractions provided at the previous two layers into one entity. Since models depend on the problem domain, the layer is divided into several independent sublayers. For the purposes of this paper, the `ivp` (the IVP representation described in the previous section) and `opt` (optimization problems) sublayers are of relevance.

At the fourth level, the two appearing sublayers are responsible for providing data structures for special types of search space decomposition used in optimization, of which only the `section` sublayer is relevant for this paper. Actual algorithms are implemented at the topmost level. UniVerMeC offers its own global optimization algorithm [7] which is employed in VeriCell for parameter identification of SOFC models (p in Eq. (1)). Additionally, external software such as IVP solvers can be interfaced at this level.

The function representation provided by UniVerMeC at the second layer plays a key role in interfacing different solvers. In Figure 4, details on `functions` are outlined. The layer consists of several abstract interfaces representing different concepts from the vicinity of mathematical functions. The main ones are `IFunction` (scalar functions) and `IVFunction` (vector functions). Both interfaces can provide additional *features* which represent concepts such as derivatives, Taylor coefficients or slopes of a function. New notions can be easily introduced by additional features attached to the abstract function representation. An important advantage of this internal structure is that the rows or single entries of a function or its Jacobian can again be viewed to be of the type `IVFunction` or `IFunction`. Thus, we can treat derivatives as *normal* functions, for example, pass them to algorithms or attach further features to them. In this way, we can represent the right-hand side of (2) such that it can be evaluated by different arithmetics and/or differentiated using various techniques. Moreover, such features as Taylor coefficients of functions can be now computed in a user-defined way and evaluated with every available arithmetic.

The uniform function representation in combination with the initial values (either

[1] that is, a layer can be skipped

intervals or floating point numbers) produces the abstract IVP problem called `IIVP` in the framework as shown Fig. 2a. The time-dependent, piecewise constant parameters $u(t)$ are described by `IVFunction` again. To interface an external solver with the framework, it is sufficient to allow it to work with problems described by the `IIVP` interface. Then, it can be used to simulate different SOFC models developed in the scope of the research project.

4 Interfacing the Solvers

In this section, we provide details on interfacing external IVP solvers with VeriCell using the library UniVerMeC. For this purpose, we outline the specifics of interfaces for each of the available solvers ValEncIA-IVP, VNODE-LP and VODE.

4.1 ValEncIA-IVP

In VeriCell, we employ the basic version of the verified IVP solver ValEncIA-IVP. It computes an enclosure of the exact solution to an IVP over a certain time interval by an algorithm derived from the Picard iteration. Basically, it relies only on enclosures of the codomain of the right-hand side of the problem at equidistant points of time specified by the user (a constant stepsize) and over intervals between them. Additionally, it requires bounds on the Jacobian of the right-hand side over the same points or time intervals. The Hessian is needed only for the sensitivity analysis which we do not consider in this paper. All three types of information can be provided using the UniVerMeC function representation directly. However, ValEncIA-IVP was designed to work as a stand-alone application and not as a library which can be accessed from external programs. Users have to specify their problem by adjusting global functions representing the right-hand side of the problem and by setting the initial values and parameters in the `main` function of ValEncIA-IVP accordingly. Besides, the solver is permanently coupled with the libraries PROFIL/BIAS [14] and Fadbad++ [21]. This makes altering the code of the solver inevitable if we are to employ it in the context of VeriCell, a dynamic environment where IVPs can be exchanged at runtime. That is why we decided to integrate ValEncIA-IVP into UniVerMeC directly (instead of just interfacing).

To minimize the necessary changes to the ValEncIA-IVP code itself and allow for easy integration of newer versions, we implemented a compatibility layer. It replaces the parts of PROFIL/BIAS and Fadbad++ APIs which ValEncIA-IVP usually expects by the corresponding parts from UniVerMeC decoupling computations from the actual data types. The interval operations are mapped onto the abstraction of an interval arithmetic employed by UniVerMeC. The actual library can be, for example, C-XSC [11], Filib++ [15], PROFIL/BIAS itself, or even the Taylor model library RiOT [9]. The derivatives are obtained through the `IDerivative` feature of `IFunction` and then mapped to the `fadbad::F` template type which ValEncIA-IVP normally uses.

Finally, we reimplemented the `main` function of ValEncIA-IVP in such as way that it can extract the initial values, the parameters and other problem or solver specific characteristics from a set based on the user supplied `IIVP` object. These adjustments allow users to exchange their problems at runtime by defining them according to the uniform problem description mentioned in Section 3 in our implementation. Solver specific options for ValEncIA-IVP are currently $\mathcal{S}_{\mathrm{VA}} = \{t_{\mathrm{end}}, s\}$, where t_{end}

is the time of integration ending and s the stepsize. These options can be extended by the user if the need arises. The implementation is not thread safe, since several global variables still have to be used.

4.2 VNODE-LP

The design principles of VNODE-LP allow for its use as an external library. The solver relies on Taylor coefficients of the right-hand side and its Jacobian. VNODE-LP accesses them through two abstract interfaces, `AD_ODE` for the right-hand side and `AD_VAR` for the variational equation. For interfacing the solver with UniVerMeC, it is only necessary to provide reimplementations of these interfaces, allowing us to obtain the required information through the uniform function representation (basically, the `ITaylorCoeff` feature). The only further difficulty is that VNODE-LP and UniVerMeC might use different interval libraries, since the library is not fixed a priori in UniVerMeC. However, a lossless conversion between the libraries is possible, since both sides represent intervals by their `double` endpoints which can be accessed. Using VNODE-LP's public interface, the initial values and other characteristics can be derived in a straightforward way from the `IIVP` object. Solver specific options are $\mathcal{S}_{\mathrm{VN}} = \{t_{\mathrm{end}}, t_0, q_{\mathrm{order}}, i_{\mathrm{sub}}, cb\}$ where t_0 and t_{end} are the integration starting and finishing times, respectively, and q_{order} is the maximum order of the Taylor expansion. The i_{sub} option can be used to subdivide the integration interval $[t_0, t_{\mathrm{end}}]$ into subintervals $[t_0, t_0 + i_{\mathrm{sub}}], \ldots, [t_0 + k \cdot i_{\mathrm{sub}}, t_{\mathrm{end}}]$ for which intermediate results are generated. Furthermore, users can specify a callback function cb which is called after the integration over each subinterval is completed. It can be used to manupulate intermediate results, parameters or solver settings and is necessary, for example, to cope with the control variables $u(t)$ which are seen as constant parameters for each subinterval but are allowed to change between them. Our side of the VNODE-LP implementation is thread safe.

4.3 VODE and ODE

Since the interfaces for floating point IVP solvers are largely standardized, the strategy for incorporating both non-verified routines ODE and VODE from the NAG library into UniVerMeC is the same; therefore we focus on VODE in the following. The most important parameters this Fortran solver expects are a pointer to the right-hand side of the IVP, the problem dimension, and possibly a pointer to the function computing the Jacobian of the right-hand side. All this information can be easily derived from the corresponding `IIVP` object. The solver settings are $\mathcal{S}_{\mathrm{VO}} = \{t_{\mathrm{end}}, i_{\mathrm{sub}}\}$, where t_{end} again is the integration ending time and i_{sub} indicates that intermediate results are required at times $t_k = k \cdot i_{\mathrm{sub}}$ within the integration interval $[0, t_{\mathrm{end}}]$. This interface is again not thread safe because the implemented wrapper routine is global in the current version.

5 Usage of VeriCell

The developed internal structure allows for a versatile use of the available SOFC models inside VeriCell. Owing to it, different algorithms, for example, floating point or interval IVP solvers, can operate with the same model representation, which reduces the amount of work for users and their potential transformation errors as well

as ensures a higher degree of interchangeability and comparability for the employed software. The following code excerpt shows an internal UniVerMeC definition for the $1 \times 1 \times 1$ model (cf. Eq. (1)) and use of the interfaces described in the previous section.

```
1  core::arith::mreal stop=19000.0;
2  core::arith::mreal veri_t0(0.0);
3  core::arith::ivector veri_x0(1,core::arith::interval(299.7053));
4  functions::IVFunction* veri_u = models::vericell_param_fun("measurements.txt");
5  core::arith::ivector veri_u0((veri_u)->eval(core::arith::rvector(1,0.0)));
6  functions::IVFunction* veri_mod = models::create_vericell_rhs_111_cpu();
7  core::arith::ivector veri_par(core::arith::ivector(
                       models::vericell_parameters(models::VERICELL_MODEL_111)));
8  ivp::details::IVPImpl ivp(*veri_mod, veri_par, *veri_u, veri_x0, veri_u0);

9  extras::interfaces::solve_ivp_vode(ivp, stop, output_vo, 1.0);
10 extras::interfaces::solve_ivp_valencia(ivp, stop, 1.0, output_va);
11 extras::interfaces::solve_ivp_vnodelp(ivp, stop, veri_t0, output_vn, 1.0, &cb, true);
```

In Lines 1–8, the `IIVP` object is defined for the simple one dimensional SOFC model. First, the integration interval is set to $[0, 19000]$ s (Lines 1,2) and the initial condition for the temperature is defined to be the point interval $[299.7053, 299.7053]$K (Line 3). We specify that the time dependent parameters $u(t)$ (declared as a UniVerMeC function) should be read from the data file `measurements.txt` in Line 4 and initialized with the values for $t = 0.0$ in Line 5. Next, the function for the right-hand side is defined (Line 6) and the parameters are assigned their values from the corresponding predefined vector (Line 7). After that, all the necessary information is there for us to be able to define the IVP-describing object `ivp`. Now it is possible to solve the IVP with the help of the available solvers (Lines 9–11) with the corresponding settings (cf. Section 4). Here, `output_` denotes the file for the output of the corresponding solver.

The environment VeriCell hides this internal structure from the user. In Figure 5, a screenshot of the VeriCell GUI is shown. In the first step, users create graphical models from the predefined components and connect them logically by arrows (black in the figure). It is also possible to load a previously saved configuration. The arrow from the element *Data* indicates that a data file is used for the parameters $u(t)$ (and not, for example, a controller), since these parameters are the only input to the element *ThermoDynamics*. Users can access the constant parameters p through a special dialog window of the thermodynamical element. After the model and the corresponding input are defined, it is possible to run an IVP solver with them, for example, VNODE-LP as in the figure. Finally, the obtained simulation results can be plotted.

An example of additional benefits offered by the increased interchangeability between floating point and interval software is validation of parameter identification results made possible in UniVerMeC. At the moment, SOFC model parameters can be identified by using a floating point solver IPOPT [23] and by our own implementation of the interval algorithm from [10]. For more details on parameter identification, see [19, 3, 12]. In UniVerMeC, it is possible to run IPOPT first and automatically assign intervals around the obtained parameter values to the verified algorithm so it can try to *verify* that the optimum is inside them. Alternatively, we can employ the interval implementation just to *filter* the search space and thus to reduce the deviation between the measured and simulated results (for example, for the cell temperature). However, the most interesting way to combine these two software kinds is, in our opinion, for *validation* of floating point results. A consistency condition for the parameter

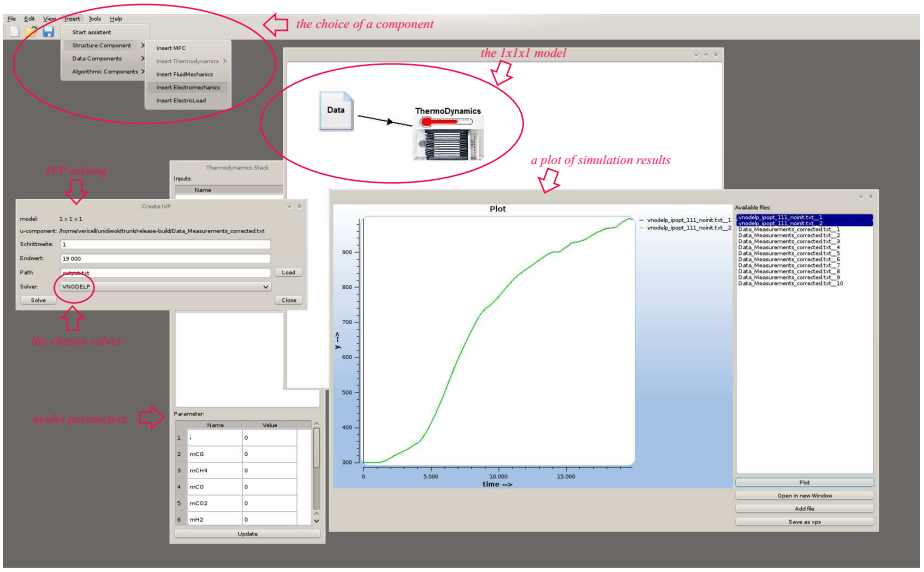


Figure 5: A screenshot of VeriCell — a GUI for the library UniVerMeC.

identification of the thermal SOFC model is that the simulated temperature values are not higher than the measured value plus 15 K and not lower than the same value minus 15 K. A temperature model with the parameter set obtained by a floating point optimizer can be simulated using a verified solver to prove that the enclosure of the simulated temperature lies inside the predefined bounds. In Figure 6, this scenario is shown for the $1\times1\times1$ model with the parameter set obtained by IPOPT: the enclosures computed by VNODE-LP are inside the interval $\pm$15K around the measured value at every point of time. The blue curve represents the upper bound of the obtained enclosures and the grey area shows the consistent region (measurements $\pm$15K). This is not true for the set of parameters obtained for the same model by the MATLAB optimizer `fminsearch`: the verified enclosure can be shown to have no intersection with the predefined bounds at certain points of time with the maximum deviation of 2 K [12]. Therefore, strictly speaking, the `fminsearch` parameter set leads to inconsistent simulation results, although this degree of deviation is not very important in a real life setting. The red curve represents the upper bound of these enclosures, the lower bound cannot be distinguished from it in the figure scale. Likewise, the deviation of 2 K is not representable in the figure scale.

In Figure 7, we compare the output temperature for SOFC models of different resolutions. For the one dimensional model, this temperature corresponds to its single state. For higher resolutions, the third and the eighth states correspond to it for the $1 \times 3 \times 1$ and $3 \times 3 \times 1$ models, respectively. Again, only the upper bounds of the obtained enclosures are shown in the figure, along with the consistent region. The simulations for the one dimensional and the nine dimensional models can be proved to be consistent, whereas the `fminsearch` parameter set leads to a slightly inconsistent output temperature for the three dimensional model (which again cannot be distinguished in the figure scale). However, the second measurable state for the

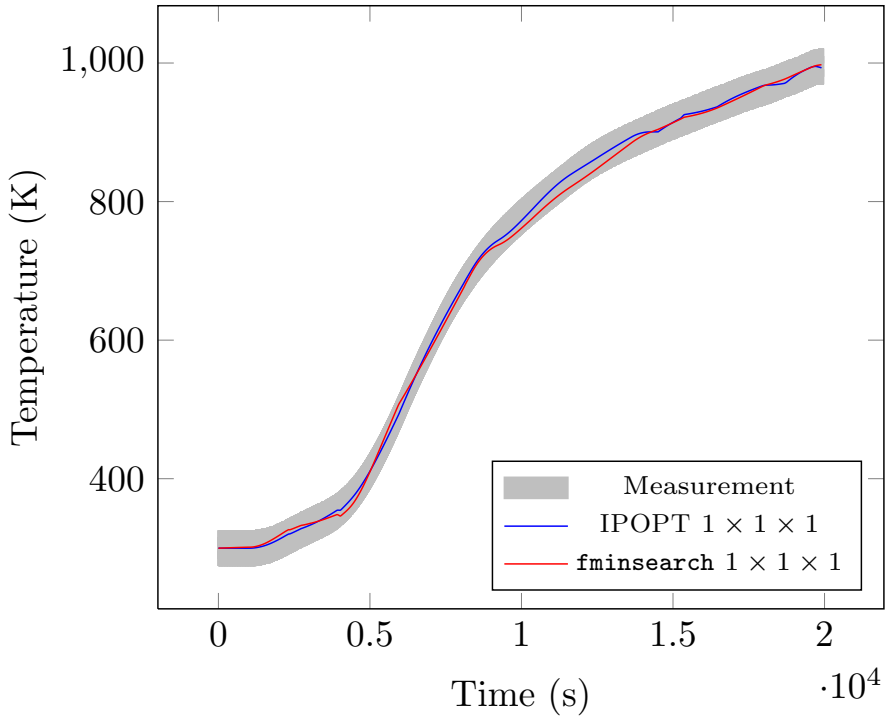


Figure 6: Upper bounds for the solution enclosures to the $1 \times 1 \times 1$ SOFC model with parameter sets from IPOPT and `fminsearch`.

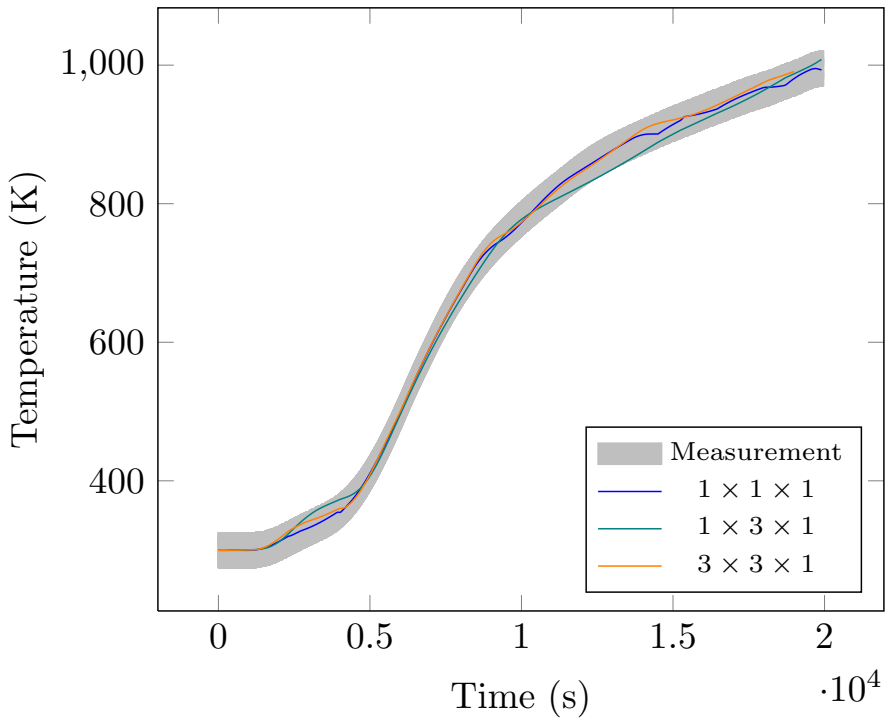


Figure 7: Simulation results for SOFC models with different resolutions. The optimized parameter sets for the $1 \times 1 \times 1$ (IPOPT) and $3 \times 3 \times 1$ (`fminsearch`) model lead to consistent trajectories for the output temperature, whereas the $1 \times 3 \times 1$ (`fminsearch`) is (slightly) inconsistent.

$3 \times 3 \times 1$ model can be shown to be inconsistent, which might be an indication that the model needs to be extended or modified to improve its correspondence to reality.

To provide graphic counterparts for those functionalities of UniVerMeC which are lacking in VeriCell (e.g. elements for optimizers) is a topic of our ongoing work.

6 Conclusions

In this paper, we presented a new software environment for testing, verification and validation of SOFC models. Owing to the internal design of UniVerMeC, it is possible to perform verified and floating point based computations using the same model definition. The interchangeability between different kinds of software allowed us to improve the correspondence between measured data and simulation as well as to validate the simulation results. Each newly developed model can be easily incorporated into the software. Additionally, further IVP solvers or optimizers can be interfaced or added to the system in a straightforward way. For user convenience, we implemented a GUI VeriCell which, however, does not reflect the whole functionality of UniVerMeC at the moment.

Our future work will be concerned with extending the GUI by further functionalities. In particular, we plan to improve the interval based parameter identification by replacing the Euler method of approximating the IVP solutions with a verified enclosure provided by VNODE-LP or ValEncIA-IVP.

Acknowledgements

This project was funded by the DFG (the German Research Foundation).

References

[1] Numerical libraries — NAG. Web page. `http://www.nag.co.uk`.

[2] E. Auer, S. Kiel, Th. Pusch, and W. Luther. A flexible environment for accurate simulation, optimization, and verification of SOFC models. In *Proc. of ASCE-ICVRAM-ISUMA, July 13–16, 2014*, Liverpool, UK, 2014. Accepted.

[3] E. Auer, S. Kiel, and A. Rauh. Verified parameter identification for solid oxide fuel cells. In *Proceedings of the 5th International Conference on Reliable Engineering Computing*, pages 41–55, Brno, Czech Republic, 2012. LITERA.

[4] R. Bove and S. Ubertini, editors. *Modeling solid oxide fuel cells*. Springer, Berlin, 2008.

[5] T. Dötschel, E. Auer, A. Rauh, and H. Aschemann. Thermal behavior of high-temperature fuel cells: Reliable parameter identification and interval-based sliding mode control. *Soft Computing*, 17(8):1329–1343, 2013.

[6] T. Dötschel, A. Rauh, and H. Aschemann. Reliable control and disturbance rejection for the thermal behavior of solid oxide fuel cell systems. In *Proc. of 7th Vienna Intl. Conference on Mathematical Modelling MATHMOD 2012*, volume 7, pages 532–537, Vienna, Austria, 2012. DOI: 10.3182/20120215-3-AT-3016.00093.

[7] E. Dyllong and S. Kiel. Verified distance computation between convex hulls of octrees using interval optimization techniques. *PAMM*, 10(1):651–652, 2010.

[8] E. Dyllong and S. Kiel. A comparison of verified distance computation between implicit objects using different arithmetics for range enclosure. *Computing*, 94:281–296, 2012.

[9] Ingo Eble. *Über Taylormodelle*. PhD thesis, Universität Karlsruhe, 2006.

[10] E. Hansen and G. W. Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 2004.

[11] W. Hofschuster and W. Krämer. C-XSC 2.0 – A C++ library for extended scientific computing. In René Alt, Andreas Frommer, R. Baker Kearfott, and Wolfram Luther, editors, *Numerical Software with Result Verification*, volume 2991 of *Lecture Notes in Computer Science*, pages 259–276. Springer Berlin / Heidelberg, 2004.

[12] S. Kiel, E. Auer, and A. Rauh. Uses of GPU powered interval optimization for parameter identification in the context of SO fuel cells. In *Proc. of NOLCOS 2013*, pages 558–563, Toulouse, France, 2013. DOI:10.3182/20130904-3-FR-2041.00169.

[13] S. Kiel, W. Luther, and E. Dyllong. Verified distance computation between non-convex superquadrics using hierarchical space decomposition structures. *Soft Computing*, 17(8):1367–1378, 2012.

[14] O. Knüppel. PROFIL/BIAS — a fast interval library. *Computing*, 53(3):277–287, 1994.

[15] M. Lerch, G. Tischler, J.W.V. Gudenberg, W. Hofschuster, and W. Krämer. `filib++`, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software (TOMS)*, 32(2):299–324, 2006.

[16] N.S. Nedialkov. VNODE-LP a validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, McMaster University, 2006.

[17] J.T. Pukrushpan, A.G. Stefanopoulou, and H. Peng. *Control of fuel cell power systems: principles, modeling, analysis and feedback design*. Springer, Berlin, 2nd edition, 2005.

[18] A. Rauh and E. Auer. Verified simulation of ODEs and DAEs in ValEncIA-IVP. *Reliable Computing*, 5(4):370–381, 2011.

[19] A. Rauh, T. Dötschel, E. Auer, and H. Aschemann. Interval methods for control-oriented modeling of the thermal behavior of high-temperature fuel cell stacks. In *Proc. of 16th IFAC Symposium on System Identification SysID 2012*, volume 16, pages 446–451, Brussels, Belgium, 2012. DOI:10.3182/20120711-3-BE-2027.00374.

[20] A. Rauh, L. Senkel, and H. Aschemann. Sensitivity-based state and parameter estimation for fuel cell systems. In *Proc. of 7th IFAC Symposium on Robust Control Design*, volume 7, pages 57–62, Aalborg, Denmark, 2012. DOI: 10.3182/20120620-3-DK-2025.00071.

[21] O. Stauning and C. Bendtsen. Fadbad++ web page. `http://www.fadbad.com/` (Accessed on 18.01.2010).

[22] K. Sundmacher, A. Kienle, H.J. Pesch, J.F. Berndt, and G. Huppmann, editors. *Molten carbonate fuel cells. Modeling, analysis, simulation, and control*. Wiley-VCH, Weinheim, 2007.

[23] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.