Software for Higher-order Sensitivity Analysis of Parametric DAEs

Moritz Schmitz[1*], Ralf Hannemann-Tams[1], Boris Gendler[2], Michael Förster[2], Wolfgang Marquardt[1], Uwe Naumann[2]

[1] Aachener Verfahrenstechnik, Process Systems Engineering, RWTH Aachen University, Templergraben 56, 52056 Aachen, Germany; *moritz.schmitz@avt.rwth-aachen.de

[2] LuFG Informatik 12: Software and Tools for Computational Engineering , RWTH Aachen University, Germany

Abstract. We introduce AC-SAMMM (The AaChen platform for Structured Automatic Manipulation of Mathematical Models), a new software infrastructure for efficient transformation and evaluation of expressions and their higher-order derivatives. We describe the way this software can be used to perform automatically the translation of a model written in an equation-oriented language like Modelica into a subset of C/C++ and the generation of the model's higher-order derivative code by algorithmic differentiation (AD) techniques. The derivatives are generated, using the *derivative code compiler* (*dcc*), an AD tool which provides source code transformation for a restricted but numerically relevant subset of C/C++. *dcc* can be applied repeatedly to its own output, to generate derivative codes of arbitrary order.

Introduction

Many engineering applications exhibit the need of modeling and simulating increasingly complex problems. Therefore, many high-level equation-based modeling languages like Modelica (www.modelica.org), gPROMS (www.psenterprise.com/gproms) and SBML (sb ml. org) have been developed to formulate mathematical models in an intuitive, equation-based representation that enhance model development and maintenance. The usage of these models in different kinds of optimization problems be- comes increasingly important.

These problems include model parameter estimation, op- timal design of experiments for parameter estimation or model structure discrimination, design optimization, model-predictive control and real-time optimization. The solution of each of these problems typically requires the evaluation of symbolic expressions involving different type and order of derivatives [6], that are in general not provided by the modeling tools.

To bridge this gap, the software package AC-SAMMM (The AaChen platform for Struc-tured Automatic Manipulation of Mathematical Models) has been jointly developed by AVT.PT (*Process Systems Engineering*) and STCE (*Software and Tools for Computational Engineering*) at RWTH Aachen University.

The aim of this project is to provide a software platform that generates robust, well-documented, and highly optimized model and derivative code by automatic manipulation of mathematical models independently of the model representation language used.

1 AC-SAMMM Introduction

Applying the AC-SAMMMis platform, by means of algorithmic differentiation [5, 9], the user is able to generate higher-order derivatives of parametric semi-explicit index-1 differential-algebraic equations,

$$\begin{aligned} \dot{x} &= f(x(t), y(t), u(p,t)), \\ 0 &= g\big(x(t), y(t), u(p,t)\big), \end{aligned} \tag{1}$$

in an automatic manner. Here, t denotes the independent (time) variable, $x(t) \in \mathrm{R}^{n_x}$ the differential or state variables, $y(t) \in \mathrm{R}^{n_y}$ the algebraic variables and $u(t, p) \in \mathrm{R}^{n_u}$ the inputs which are parameterized by some parameters $p \in \mathrm{R}^{n_p}$ for convenience and without loss of generality.

Additional drivers are provided that support the efficient evaluation of the generated derivatives and the original mathematical model. The generated code is compiled into a dynamic link library or a shared object that can be accessed by third- party software (e.g. optimization and simulation software) via an C++ interface.

As illustration of the importance of fast and accurate higher-order derivatives, we consider *optimal control problems* and their solution by means of control vector parameterization (cf. Brusch [2]).

In optimal control, the user has to control a dynamic process such that an objective function is minimized or maximized and usually some process constraints have to be satisfied. We consider the following class of Mayer-type optimal control problems (**OCP**), already discretized by means of control vector parameterization, resulting in the finite-dimensional nonlinear program (NLP).

$$\min_{p,t_f} \Phi\left(x(t_f), y(t_f)\right) \tag{2}$$

$$s.t. \quad \dot{x} = f(x, y, u(p,t)), \;\; t \in \left[t_0,\, t_f\right] \tag{3}$$

$$0 = g(x(t), y(t), u(p,t)) \tag{4}$$

$$0 = x(0,p) - x_0(p) \tag{5}$$

$$0 \geq h(x(t_k), y(t_k), u(p,t_k)), k = 1, \dots, N \tag{6}$$

$$0 \geq e(x(t_f), y(t_f), u(p,t_f)) \tag{7}$$

This NLP is typically solved by gradient-based optimization methods such as SQP or interior point solvers. In general, these solvers need at least first-order derivatives of the objective function and the constraints and sometimes also second-order derivatives to form the corresponding Hessian of the Lagrangian. Therefore, an efficient and accurate calculation of these derivatives is fundamental when it comes to fast and accurate numerical calculations.

The paper is organized as follows. First, the typical workflow of AC-SAMMM is presented. Then, we explain the design of the `dcc`-generated derivative codes and show how they are called in an efficient way. In Section 4, details about the workflow and the functionality of the ESO are presented. Section 5 discusses an application of AC-SAMMM to a small OCP. Section 6 concludes the paper and gives directions for future development.

2 Typical Workflow in AC-SAMMM

Today, AC-SAMMM is able to translate mathematical models written in flat Modelica code into C- code using `Mof2C-` (a parser developed at AVT.PT) and to generate derivative code of arbitrary order by applying the AD-tool `dcc` [12] (developed at STCE) to the generated code.

Flat Modelica prescribes code where all object-oriented and hierarchical features of Modelica are eliminated ('flattened'). Though not really standardized in the language specification, the flattening process is considered to be part of all executable Modelica simulation environments (see [8, Chapter5]). C- is a small subset of the programming language C/C++. Additional algebraic manipulations will be applied and a highly optimized dynamic library will be obtained.

After the application of AC-SAMMM the original model information and additional derivative information (restricted to second- order derivatives today) are stored in the dynamic library. The access to this information is managed by a so-called *Equation Set Object* (ESO) which is an instance of a standardized C++ class.

The AC-SAMMM ESO is slightly modified variant of the ESO specified by the CAPE-OPEN organization in [11]. This ESO encapsulates all calls to the model residuals and related derivative functions and can be interfaced to different applications. The typical workflow of generating such an ESO is shown in Figure 1.

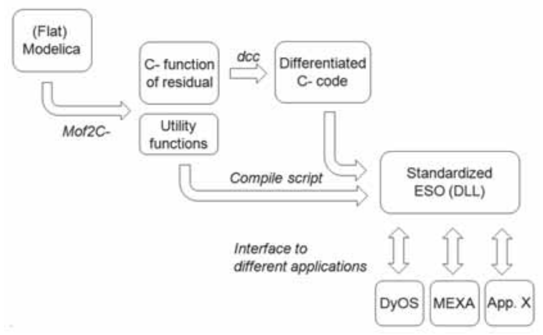

Figure 1. Workflow of AC-SAMMM. The flat Modelica code is translated to C- by Mof2C-. The C- files contain the residual function and some utility functions providing amongst others the variable and parameter names and the initial values for the variables (if algebraic variables are existent, their initial values are not necessarily consistent with the algebraic equations). In a second step, the residual function is differentiated by dcc. Finally all generated functions are compiled into a dynamic library defining the ESO-functions. The ESO defines the interface to third-party applications.

3 The Derivative Code Compiler `dcc`

For the reasons stated in Section 1, most optimization tools need higher-order derivatives of the dynamic model. AD source code transformation provides derivative code of arbitrary order. The derivative values computed by this code are accurate up to machine accuracy as opposed to the approximated values computed by finite differences.

Derivatives can be generated in the tangent-linear or in the adjoint mode. We define $z = (x, y)$ and consider

$$F(z,\dot{z},u) = \begin{pmatrix} \dot{x} - f(x,y,u) \\ g(x,y,u) \end{pmatrix} = 0$$

The software package AC-SAMMM uses the derivative code compiler `dcc` to generate the derivative code of an input code in tangent-linear or in adjoint mode. The input code is given in a subset of the programming language C/C++ called C-. `dcc` is able to use its own output again as input what allows the generation of derivative code of arbitrary order. The first-order tangent-linear model is defined as

$$\mathbb{R}^{n_x+n_y} \ni F^{(1)} \equiv F'(z,\dot{z},u) \begin{pmatrix} z^{(1)} \\ \dot{z}^{(1)} \\ u^{(1)} \end{pmatrix} \tag{8}$$

where $\left(z^{(1)},\dot{z}^{(1)},u^{(1)}\right)^T \in \mathbb{R}^{2(n_x+n_y)+n_u}$

Given the implementation of F in C- `dcc` generates code that takes $(z^{(1)}, z^{\cdot(1)}, u^{(1)})^T$ as input and computes $F^{(1)}$ with computational costs of about twice as high as the computational costs of F. The corresponding C++ implementation that is part of the ESO (see Section 4) has the following signature, given in Listing 1, where the prefix t_1 stands for *first-order tangent-linear* and has the same meaning as the superscript $^{(1)}$.

```
void eval_t_1 residuals ( double* t1_z,
double* t1_der_z, double* t1_der_u,
double* t1_residuals )
```

Listing 1. ESO-function calling first-order tangent-linear model.

The output of the function representing $F^{(1)}$ is t1_residuals . All other arguments are the user's input. The actual values of the states and the parameters are members of the class providing this implementation and are known implicitly by the method.

The first-order adjoint model's output $(z_{(1)},\dot{z}_{(1)},u_{(1)})^T \in \mathbb{R}^{2(n_x+n_y)+n_u}$ is defined as

$$\begin{pmatrix} z^{(1)} \\ \dot{z}^{(1)} \\ u^{(1)} \end{pmatrix} \equiv \begin{pmatrix} z^{(1)} \\ \dot{z}^{(1)} \\ u^{(1)} \end{pmatrix} + \left(F'(z,\dot{z},u)\right)^T F_{(1)} \tag{9}$$

where the vector $(z_{(1)},\dot{z}_{(1)},u_{(1)})^T$ has to be given as input and will be overwritten by the new output value. Note that we use parenthesized superscripts to denote tangent- linear projections and parenthesized subscripts to mark adjoint projections. The ESO- implementation is analogous to that of the tangent-linear model with prefix *a_1* corresponding to subscript $_{(1)}$.

The computational costs of the adjoint model are about six times higher than the costs for evaluating F (for an estimation of the computational costs, see [5]).

Nevertheless, in case of a matrix vector product where the matrix is the transposed Jacobian, the adjoint model is the best choice. The only alternative to compute this matrix vector product would be the accumulation of the whole Jacobian to be able to transpose it afterward.

The second-order tangent over adjoint model's output $(z_{(1)}^{(2)},\dot{z}_{(1)}^{(2)},u_{(1)}^{(2)})^T \in \mathbb{R}^{2(n_x+n_y)+n_u}$ is defined as

$$\begin{pmatrix} z_{(1)}^{(2)} \\ \dot{z}_{(1)}^{(2)} \\ u_{(1)}^{(2)} \end{pmatrix} \equiv \begin{pmatrix} z_{(1)}^{(2)} \\ \dot{z}_{(1)}^{(2)} \\ u_{(1)}^{(2)} \end{pmatrix} + < F_{(1)}, F''(z,\dot{z},u) \begin{pmatrix} z^{(2)} \\ \dot{z}^{(2)} \\ u^{(2)} \end{pmatrix} > + F'(z,\dot{z},u)^T F_{(1)}^{(2)}$$

where the second term is the projection of the $((n_x + n_y) \times (2(n_x + n_y) + n_u) \times (2(n_x + n_y) + n_u))$ Hessian tensor $F''(z,\dot{z},u)$ in directions $F_{(1)} \in \mathbb{R}^{n_x+n_y}$ and where $(z_{(1)},\dot{z}_{(1)},u_{(1)})^T \in \mathbb{R}^{2(n_x+n_y)+n_u}$. The ESO-method that accesses the dcc-generated second-order model is shown in Listing 2. To exploit structural facts of the Jacobian and Hessian we use coloring algorithms to find possibilities to compress the Jacobian and Hessian [14, 15].

```
void eval_t_2_a1 residuals ( double* t2_z,
 double* a1_z, …., double* a1_f,
 double* t2_a1_f )
```

Listing 2. ESO-function calling second-order tangent over adjoint model.

4 AC-SAMMM in Practice

The above mentioned Jacobian compression techniques are used in the AC-SAMMM Jacobian and Hessian driver routines and are automatically executed if the user calls one of the ESO-methods to access derivatives. For example, the ESO-method

```
void  get_jacobian_values ( long len ,
    long* indices, double* jacobian) ;
```

calls the first-order tangent-linear code (see assignment 8) with optimal seeding vectors $(z_{(1)},\dot{z}_{(1)},u_{(1)})^T$ to get the specified Jacobian entries with minimal computational effort. The argument *len* defines the length of the output array *jacobian*.

The entries in *indices* define the relative indices of the wanted non-zero elements in the jacobian matrix arranged in row-major format $\left(\frac{\partial F}{\partial z}\ \frac{\partial F}{\partial u}\right)$.

The ESO defines the C++ interface to all client-applications using AC-SAMMM. The AC-SAMMM-ESO definition is based on the CAPE-OPEN ESO interface and contains some additional functions. The ESO is intended to cover at least the required functions to solve a differential-algebraic system and additional functions to handle higher-order derivatives.

So far AC-SAMMM can only be used on Microsoft Windows platforms, but AC-SAMMM will be adapted to deal with multiple platforms such as *Microsoft Windows*, *Linux/Unix* and *Mac OS*, soon. In order accomplish this, AC-SAMMM is developed using CMake [7]. The way starting from a dynamic model coded in Modelica to the dynamic library including the higher-order derivatives and the original model information is completely automated (see Figure 1). The user calls a script that takes as argument the name of the flat-Modelica model. The flat-Modelica parser `Mof2C-` creates the C-code of the model. Beside the residual function, that is presented in detail in Section 5, other important files are generated, amongst others the *block residual function*.

The block residual function provides a signature with an index-set to define the indices of the residuals to evaluate. This offers an important gain in computational time if only a small subset of the residual vector should be evaluated. This is the case if, for example, a block-decomposition routine for the determination of consistent initial values is used. Additionaly, `dcc` uses operator overloading techniques in combination with the propagation of bit patterns to determine the sparsity pattern of the Jacobian.

The next fully automated step during the script call is the call of dcc. The first- and second-order derivatives of the residual-model are generated and will be used later on to define the ESO-functions (see Listing 1 and Listing 2). AC-SAMMM is restricted to second-order derivatives (tangent over adjoint mode) but could be easily modified to provide even higher-order derivatives. In the final step, the dynamic library is created using a C++ compiler (which has to be installed on the user's PC). The hereby generated model library can then be loaded dynamically (see Listing 4) into the AC-SAMMM-ESO providing amongst others the efficient drivers for derivative evaluation up to second order by exploiting sparsity.

Some examples of ESO-methods have been presented so far. For a complete description on all ESO-methods we refer to the AC-SAMMM manual [13].

5 Case Study

In the introduction we mentioned the problem class *optimal control problems*. In this section we deal with a problem belonging to this class that will serve as a first proof of concept.

We consider the illustrative and very simple problem of a car that has to travel a fixed distance in minimal time. In adddition, the car has to start at rest and its final velocity has to be zero. The control parameter is its acceleration *u*(*t*) and with parameter α.

$$
\begin{array}{cc}
\min\limits_{u(t)} t_f & \\
\dot{x} = v & x(t_f) > x_{t_f} \\
\dot{v} = u - \alpha v^2 & v(t) \leq v_{max} \\
x(0), v(0) = 0 & -1 \leq u(t) \leq 1 \\
v(t_f) = 0 &
\end{array}
$$

One can easily verify that this problem formulation fits the generic form **OCP**, if an additional state representing the time is introduced. The corresponding (flat) Modelica code for the dynamic model is given in Listing 3.

```
model  car
parameter Real accel = 2;
parameter Real alpha = 0.0025;
Real ttime; Real velo; Real dist;
equation
der(dist) = velo;
der(velo) = accel – alpha *pow(velo,2);
der(ttime) = 1;
end car;
```

Listing 3. Flat Modelica code for case study.

The Modelica model represents only the dynamic model that defines neither an objective function nor additional constraints, so the control *accel* = *u*(*t*) is defined as a constant parameter. The core-capacity of AC-SAMMM is the manipulation of dynamic models of the form (1). Hence, we will first explain the treatment of manipulating the dynamics before dealing with the entire optimization problem.

The script call that executes the operations mentioned in Section 4 has to be entered in a command-line interface being able to interpret BASH-commands (i.e. cygwin (www.cygwin.com) for Windows platforms). If the model's name is *car.mof*, then the script is being called by the command `acsammm car`. The intermediate C- functions generated by `Mof2C-` contain all the model information provided by the Modelica model. The function representing the residual function (shown in Listing 4) will be processed by `dcc`.

```
void res (double * yy, double * der x,
  double * x, double * , int &n_x, int &n_p)
// $ad indep x p; ……// $ad dep yy
{  // …….// scalar equation 1
acs pow(var_velo, 2, var pow0);
yy[i_E] = der_var_velo –
    (par_accel – par_alpha * var_pow0);
i_E = i_E + 1; / / …… }
```

Listing 4. The residual function in C-.

The comments that appear after the signature are special comments for dcc to indicate which variables are dependent (here *yy*) and should be differentiated with respect to the independent ones (here *x* and *p*). Furthermore one can recognize that *yy* corresponds to the residual-function representing the dynamic model.

AC-SAMMM overloads the basic mathematical functions sqrt(x), x^a, (x/y), log(x) and exp(x) in order to increase robustness and/or efficiency of their numerical evaluation (see *acs_pow*(x,a,y) in the above code fragment). An example is the mirroring of the square root at the origin. This is useful, for instance, when $x > 0$ passes to $x < 0$ immediately after a switching point, in case of a hybrid system. In order to detect a switching point precisely, a switching structure detection algorithm has to pass this point without switching being accomplished [3].

As a result of the script call the dynamic libraries called `car.dll` (in release mode) and `car_d.dll` (in debug mode) are created. The intermediate C++ files are still available to the user but not needed to any further application of the model library.

The generated libraries are written in a neutral format so that they are not application- specific and can easily be exploited for a variety of purposes. We illustrate the application in the case of OCP, which can be resolved by means of direct single shooting, which has originally been introduced by Brusch [2]. The basic idea is to substitute the control vector by an approximation (typically piecewise-constant or -linear), and to relax the path constraints on a grid $t_0 < t_1 < ... < t_n = t_f$. This way the infinite-dimensional optimal control problem is approximated by a finite-dimensional nonlinear program (NLP). For this purpose we introduced a new C++ class called *MetaESO*. The aim of the MetaESO class is to define the constraints, the controls and the underlying time grids as input for the integrator. In addition, the user can define the sensitivities to be calculated. In summary, the user can define a restricted multistage OCP using the AC-SAMMM- generated dynamic library as dynamic model within this class (at present only: explicit switching times, simple box-constraints).

Using an integrator and an optimizer, the infrastructure is suited to solve a restricted class of multistage optimal control problems. The integrator used is NIXE (Hannemann et al. [6]). NIXE implements the extrapolated linearly-implicit Euler discretization for the solution of parametric differential-algebraic initial value problems (given by equations (3) - (5) of formulation **OCP**) and computes higher-order sensitivities by an efficient modified higher-order discrete adjoint approach or by forward sensitivity analysis. We show very briefly how to use AC-SAMMM as embedded dynamic model server, how to define the OCP and how to call the integrator NIXE. We omit details concerning the optimizer application and show the basic usage in Listing 5.

```
ACSAMMM_Eso* CarEso = new ACSAMMM_Eso( car );
MetaEso metaEso* = new MetaEso( );
metaEso->AddPhysicalStage( CarEso, 0.0, 1.0);
```

Listing 5. Dynamic loading of the model-library *car* in AC-SAMMM and usage of MetaEso class.

The second and third argument of *AddPhysicalStage()* represent the start and final time for the integration. As problem (10) has free final time, we had to reformulate:

$$x(t) = \Delta t\, f\big(x(t'), y(t'), u(p, t')\big), \qquad t' \in [0,1)$$
$$0 = g\big(x(t'), y(t'), u(p, t')\big),$$

where $\Delta t = t_f - t_0$ is a parameter that can be controlled.

The constraints are defined using the *MetaESO* function *AddConstraint(stage Index, name, time, lB, uB, lagrMult)* as shown in Listing 6. The input *lagrMult* represents the value of the Lagrange multiplier corresponding to the constraint. This input could be delivered by an optimizer.

```
MetaEso->AddConstraint(0,velo,1.0,0,0,lagrMult);
MetaEso->AddControl (0,accel,piecewiseLinear,
             grid, parameters , -1.0 , 1.0 );
```

Listing 6. *MetaESO* : Constraints and Controls

In the same way the control parametrization of the acceleration could be the output of an optimizer. So the fifth argument of *AddControl*(*stageIndex*, *name*, *typeControl*, *grid*, *parameters*, *lB*, *uB*) is defined by the optimizer. The constraints define the states where sensitivities should be calculated. They are calculated with respect to the control-parametrization. The second-order sensitivities are calculated with respect to the Lagrangian function. Finally, as shown in Listing 7, the *MetaESO* object can be transferred to the integrator calculating the specified sensitivity information. More details about definition of related OCP see [13].

```
SecondOrderReverseHandler∗ sorHandler =
  new SecondOrderReverseHandler ( MetaEso );
sorHandler -> solve( );
```

Listing 7. *MetaESO* handler for second-order derivatives.

An interesting case study serving as a benchmark is a large-scale nonlinear system of about 2000 stiff DAEs [4]. It models the load change of an polymerization process. For real-time optimization (cf. Würth et al. [16]), about 40 second-order parameters and 160 first-order parameters of the model have to be computed. We measured the generation- time of the higher-order derivatives as well as the compile- and evaluation-times. All computations were performed on a Core2-Quad PC running Windows 7 on a 2.66 GHz CPU. The size of the dynamic library in release mode is about 6 MB.

Code generation time	Size of generated code	Compile time		Computation time of Hessian of Lagrange.
		with opt.	without opt.	
15 min	20 MB	hours	seconds	≈ 3 min.

Table 1. Computation effort for case study with 2000 stiff DAEs.

6 Summary and Outlook

We presented a platform for automatic algorithmic differentiation of mathematical models, which exhibits an object-oriented (extensible) interface that suits the needs to solve a differential-algebraic system. Included are `Mof2C-`, a Modelica to C code parser and dcc, the derivative code compiler for automatic differentiation.

A prototype-driver for multistage OCP exists serving as an interface between AC-SAMMM and NIXE. This interface will be developed further in future versions of AC-SAMMM. Also, further reductions of compile- and run-times can be expected. A major concern of the future development of AC-SAMMM is the proper treatment of hybrid systems [1], especially enabling discontinuity-locking and the handling of arbitrary complex switching conditions.

References

[1] Barton, P. I., and Lee, C. K. (2002). Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Trans. Model. Comput. Simul.*, 12, 256–289.

[2] Brusch, R. G. (1974). A nonlinear programming approach to space shuttle trajectory optimization. *Journal of Optimization Theory and Applications*, 13(1), 94–118.

[3] Cellier, F. E., and Kofman, E. (2006). *Continuous System Simulation*. Springer, New York.

[4] Dünnebier, G., van Hessem, D., Kadam, J., Klatt, K.-U., and Schlegel, M. (2005). Optimization and control of polymerization processes. *Chemical Engineering Technology*, 28(5), 575–580.

[5] Griewank, A., and Walther, A. (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Soc. for Industrial and Applied Math. (SIAM).

[6] Hannemann, R., Marquardt, W., Gendler, B., and Naumann, U. (2010). Discrete first- and second-order adjoints and automatic differentiation for the sensitivity analysis of dynamic models. In *Procedia Computer Science*, volume 1, pages 297–305.

[7] Martin, K., and Hoffman, B.(2003). *Mastering CMake: A Cross-Platform Build System*. Kitware Inc..

[8] Modelica Association, Linköping, Sweden. (2010). *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification. Version 3.2*, March 2010.

[9] Naumann, U. (2011). *The Art of Differentiating Computer Programs*. SIAM, 2011. To appear.

[10] Naumann, U., Schenk, O., Simon, H., and Toledo, S., editors. (2009). *Combinatorial Scientific Computing.* number 09061 in Dagstuhl Seminar Proceedings, Wadern, Germany. Leibnitz-Zentrum für Informatik.

[11] Pantelides, C., Keeping, B., Bernier, J., and Gautreau, C. (1999). *Open interface specification numerical solvers*. Technical Report CO-NUMR-EL-03 Version 1.08, CAPE-OPEN, 1999.

[12] Schanen, M., Förster, M., Gendler, B., and Naumann, U. (2011). Compiler-based Differentiation of Numerical Simulation Codes. In *ICCGI 2011, The Sixth International Multi-Conference on Computing in the Global Information Technology*, pages 105–110. IARIA.

[13] Schmitz, M., Gendler, B., and Hannemann, R. *Introduction to AC-SAMMM: A tutorial installing, and using AC-SAMMM*. RWTH Aachen, AVT.PT Process Systems Engineering, Aachener Verfahrenstechnik, Turmstraße 46, 52062 Aachen, to be made accessible.

[14] Varnik, E. (2011). *Exploitation of Structural Sparsity in Algorithmic Differentiation*. PhD thesis, RWTH Aachen University, Aachen, Germany, submitted.

[15] Varnik, E., and Naumann, U. (2009). What Color is the Non-Constant Part of Your Jacobian? In *[10]*, 2009, Extended Abstract.

[16] Würth, L., Hannemann, R., and Marquardt, W. (2011). A two-layer architecture for economically optimal process control and operation. *Journal of Process Control*, 21(3), 311–321.

Submitted: July 2011 (ASIM STS Winterthur)
Accepted SNE: July 20, 2012