Abstract

The Jülich-Aachen Dynamic optimization Environment (JADE) is employed for computing first- and second-order parameter sensitivities of a metabolically and isotopically non-stationary biochemical network model. Based on a Modelica representation of the model, code generation, algorithmic differentiation and first- and second-order adjoint sensitivity analysis are employed for computing the gradient and the Hessian of a parameter estimation objective function. In particular, we use composite adjoints, an extension of the classical adjoint sensitivity analysis, and a numerical integrator based on the extrapolated linearly-implicit Euler method. Therewith, the $116 \times 116$-Hessian of the objective function with respect to 116 model parameters can be computed at the cost equivalent to only 18 objective function evaluations, while computing the same Hessian with the cheapest finite-difference formula would require 6845 evaluations of the objective function.

Keywords: biochemical network model; parameter sensitivities; automatic differentiation

1 Introduction

Kinetic-based modeling is the method of choice for unraveling complex interactions in living microorganisms [8]. Only this approach allows to analyze the response of organisms to extracellular stimuli, such as changes in the substrate concentration. Moreover, industrial processes are typically run in cultivation modes, in which the intracellular metabolism cannot be assumed to be in a stationary state. Metabolically non-stationary network models include rate laws for the enzyme catalyzed reactions, and the corresponding model equations depend on several kinetic parameters. The rate laws also include regulatory effects, i.e. activation and inhibition by other metabolites, which increases the overall complexity of the network. Kinetic models are normally calibrated using measured intracellular metabolite concentrations. However, the ratio between the number of unknown parameters and the quantity of available measurement data is often insufficient. Consequently, the kinetic parameters are poorly determined or even non-identifiable on the basis of such data.

This limitation can be overcome by combining classical kinetic modeling with an isotope-labeling approach ([11], [3]). In this approach, experiments are performed with a specifically $^{13}$C-labeled substrate instead of the slightly lighter, naturally $^{12}$C-labeled substrate. $^{12}$C as well as $^{13}$C-atoms are distributed through the reaction network and form specific labeling signatures in the involved metabolites. These signatures, so-called isotopologues, consist of differently many heavier (labeled) and lighter (naturally labeled) carbon atoms, and can be quantified using the LC-MS measurement technique [12]. Hence, the use of labeled substrates increases the amount of data for each metabolite proportional to its number of carbon atoms. However, the model dimensions are strongly increased. The extended model requires increased computational resources not only for solving the forward problem, but also for determining gradient and...
Hessian information for efficiently solving the inverse parameter estimation problem.

2 Biochemical Network Model

The combined metabolically and isotopically non-stationary modeling approach is illustrated with an example network of *Escherichia coli* [2]. The biochemical network covers glycolysis and the pentose phosphate pathway. The model involves 28 metabolites (thereof 8 co-metabolites) and 28 reactions (thereof 8 effluxes), as illustrated in Figure 1.

The equations for the kinetic rates, \( r \), the values of the stoichiometric constants, \( p_{\text{stoich}} \), and of the kinetic parameters, \( p_{\text{kin}} \), and the initial metabolite concentrations, \( c_0 \), are taken from the same publication [2]. The model is extended from the metabolically non-stationary case to the metabolically and isotopically non-stationary case by transforming the differential equations, that describe the change of metabolite concentrations, \( c \), over time (Equation 1), into sets of differential equations for the so-called cumomers, \( m \) (Equation 2).

\[
\frac{dc}{dt} = f(c, r, p_{\text{stoich}}) \\
\begin{align*}
\frac{dr}{dt} &= g(c, p_{\text{kin}}) \\
c(0) &= c_0 \tag{1}
\end{align*}
\]

A cumomer can be interpreted as a molecule fragment that is fully labeled to a specified degree ([13], [14]). The cumomer, e.g., \( m_{\text{#1x1}} \) of a metabolite, \( m \), with three carbon atoms includes the four differently labeled species \( m_{\text{#1x1}} = \sum_{ij\in\{0,1\}} m_{\text{#ij1}} \), namely \( m_{\text{#010}}, m_{\text{#110}}, m_{\text{#011}} \) and \( m_{\text{#111}} \), where the digits 0 and 1 denote the isotopes \( ^{12}\text{C} \) and \( ^{13}\text{C} \), respectively. The concentration of a cumomer fraction is defined as the sum of the concentrations of all corresponding species. In particular, the concentration of the cumomer \( m_{\text{#xxx}} \) is the absolute metabolite concentration, \( c \). A metabolite with \( a \) carbon atoms has \( 2^a \) cumomers in total. The formulation of cumomer balance equations requires structural information on: (1) the underlying metabolic network model, i.e. all participating enzymatic reaction steps, (2) the carbon atom transitions for each of these steps (see Figure 2 for an example), and (3) the kinetic mechanisms [11].

\[
\frac{dm}{dt} = f(m, r, p_{\text{stoich}}) \\
m(0) = m_0 \tag{2}
\]

The cumomer balances in Equation 2 are combined with the original kinetic equations from Equation 1.

The vector \( c \), containing all metabolite concentrations, is a subset of the vector \( m \), containing all cumomer fractions \( m_{\text{#ijk}} \) with \( i, j, k \in \{1, x\} \) of all metabolites. The initial values of the algebraic variables, \( r \), are determined such as to fulfill the algebraic equation, \( g \).

Realistic models, e.g., of the central carbon metabolism, have around 30-40 metabolites, 50-60 reactions and 30-40 regulatory relations leading to model dimensions of 1,000 to 10,000 equations. Moreover, Equation 2 is typically stiff, dense and highly non-linear.

![Figure 2: Carbon atom transition of a reaction that converts D-fructose-1,6-bisphosphate (FBP) into glyceraldehyde-3-phosphate (GAP) and dihydroxyacetone-phosphate (DHAP). The lines describe transitions of individual carbon atoms from the substrate to the product.](image-url)
Figure 1: Biochemical network of *E. coli* including the glycolysis (orange) and the pentose phosphate pathway (red). The metabolites (rectangles) are converted by reactions (rhombi). Additional lines show regulatory interactions: activation (green lines), inhibition (red lines) and co-metabolite coupling (dashed lines, yellow metabolites).

3 JADE

The biochemical network model from the previous section has been implemented in Modelica and tested with Dymola. However, Dymola does not provide capabilities for higher-order sensitivity analysis, which are essential for many engineering tasks such as parameter estimation, optimal experimental design, optimal control and dynamic real-time optimization (DRTO). This gap will be closed by the *Jülich Aachen Dynamic Optimization Environment* (JADE), a new research program that sustains ongoing collaborations between Aachener Verfahrenstechnik – Process Systems Engineering (AVT,PT), the Jülich Biotechnology Institute (IBG-1), and Software Tools for Computational Engineering (STCE). AVT,PT and STCE are both chairs at RWTH Aachen University and IBG-1 belongs to the Forschungszentrum Jülich. The JADE concept includes a software infrastructure for sensitivity analysis of differential-algebraic equation systems.

This publication addresses a prototypical task within the JADE framework, the determination of parameter sensitivities of a residual function for estimating unknown model parameters. The biochemical network example is taken as an example, but the presented software infrastructure works for a wider class of Modelica models, without discontinuous elements, i.e. without “if”- and “when”-assignments. A software infrastructure is presented, that provides an easy-to-use integrated solution for determining the required
First- and second-order parameter sensitivities of a metabolically and isotopically non-stationary …

\[ r_{pgi} = \frac{\max_{x} \left( c_{\text{G6P}} - c_{\text{G6P}} \right)}{\text{km}_{\text{G6P} \cdot \text{pgi}}} \left( 1 + \frac{c_{\text{ATP}}}{\text{km}_{\text{ATP} \cdot \text{pgi}}} \right) + c_{\text{G6P}} \]

\[ r_{pgk} = \frac{\max_{x} \left( c_{\text{ADP}} \cdot c_{\text{PGP}} - c_{\text{ATP} \cdot c_{\text{ADP}} \cdot \text{PGP}} \right)}{\text{km}_{\text{ADP} \cdot \text{pgk}}} \left( 1 + \frac{c_{\text{ATP} \cdot c_{\text{PGP}}} + c_{\text{G6P}}}{\text{km}_{\text{ADP} \cdot \text{pgk}}} \right) + c_{\text{PGP}} \]

first- and second-order derivatives.

**Workflow**

The workflow for computing sensitivity information can be divided in three layers (see Figure 3 for a schematic sketch):

1. A so-called equation set object (ESO), an instance of a C++ class which provides data and methods related to the model.

2. A Meta ESO object, an instance of a C++ class which assembles one ESO or, in the case of multistage models, several ESOs and information about the parametrization of the model (we refer to [9, 10] for details on multistage problems).

3. Drivers for the NIXE integrator [5], a numerical solver for (adjoint) sensitivity analysis of DAEs, based on the information assembled in the Meta ESO, to carry out sensitivity analysis tasks.

Currently, flat Modelica models are translated into a subset of the C language, referred to as C-, by means of the Mof2C- application. In flat or flattend Modelica models, all object-oriented features are removed by the expansion of all sub-models and their connections. In particular, a flat Modelica model contains no sub-model, it has a “flat hierarchy”. A residual function of the DAE is created to be differentiated by means of algorithmic differentiation in form of the derivative code compiler (dcc) [7], an AD tool relying on semantic source code transformation. On Windows platforms, the source code, generated by Mof2C- and dcc is compiled into a dynamic link library. Figure 4 shows a typical workflow within the JADE framework.

**4 Results**

We present first- and second-order adjoint sensitivity computations for the biochemical network model from section 2. The model is formulated in Modelica without using discontinuous elements. It belongs to the class of smooth semi-explicit index-1 differential algebraic equations of the type of Equations 5 to 7.

\[ \dot{x}(t, p) = f(x(t, p), y(t, p), p), \quad t \in [t_0, t_f], \quad (5) \]

\[ 0 = g(x(t, p), y(t, p), p), \quad t \in [t_0, t_f], \quad (6) \]

\[ x(t_0, p) = x_0, \quad (7) \]

Here, \( x(t, p) \in \mathbb{R}^{n_x} \) and \( y(t, p) \in \mathbb{R}^{n_y} \) denote the vectors of differential and algebraic state variables, \( p \in \mathbb{R}^{n_p} \) is the parameter vector, \( f \) and \( g \) denote the differential and algebraic equations, respectively, \( x_0 \in \mathbb{R}^{n_x} \) is the vector of initial values and \( t_0 \) and \( t_f \) are the initial and final times, respectively.

The model comprises 1488 state variables, thereof 683 differential and 805 algebraic, as well as 337 parameters, thereof 116 relevant for a typical parameter
estimation. The model is sparse in that the Jacobians of \( f \) and \( g \) with respect to \( x, y \) and \( p \) have in the sum only 9121 nonzero entries. The initial time is set to \( t_0 = -20 \) in order to simulate the system in a stationary state before a concentration pulse is applied at \( t = 0 \), and the final time is \( t_f = 40 \).

For the purpose of parameter estimation we need to compute a least-squares residual, as well as its gradient and Hessian. Let \( y_{out}(t, p) = (y_1(t, p), \ldots, y_{n_{out}}(t, p)) \in \mathbb{R}^{n_{out}}, \ i_j \in \{1, \ldots, n_y\}, \ j = 1, \ldots, n_{y_{out}} \) denote the vector of measured variables, which is in the present example a subset of the algebraic variables. For the residual we consider a finite time series \( t_1 < t_2 < \cdots < t_N \) and a matrix of corresponding measurements \( \tilde{Y} = (\tilde{y}_{ij}) \in \mathbb{R}^{N \times n_{out}} \). With scalar weights \( \sigma_{ij}, i = 1, \ldots, N, j = 1, \ldots, n_{y_{out}} \), the parameter estimation objective function has the following form:

\[
\Phi(p) = \phi(y_{out}(t_1, p), y_{out}(t_2, p), \ldots, y_{out}(t_N, p)) := \sum_{i=1}^{N} \sum_{j=1}^{n_{out}} \sigma_{ij} (y_{out,j}(t_1, p) - \tilde{y}_{ij})^2. \tag{8}
\]

We assume measurements to be available for \( n_{y_{out}} = 103 \) output variables every 0.5 seconds, starting from \( t_1 = 0 \) to \( t_N = t_{81} = 40 \). As real measurements are currently not available, synthetic data \( \tilde{Y} = (\tilde{y}_{ij}) \in \mathbb{R}^{81 \times 103} \) were generated by adding normally distributed noise with a standard deviation of 10% to the nominal values. The weights are chosen as:

\[
\sigma_{ij} = \frac{1}{0.01 + \tilde{y}_{ij}} , \ i = 1, \ldots, 81, \ j = 1, \ldots, 103,
\]

The summand 0.01 in the denominator is introduced for avoiding division by zero in the case \( \tilde{y}_{ij} = 0 \) and to reduce the impact of small-valued measurements.

Let \( p_{est} \in \mathbb{R}^{n_{p_{est}}} \) denote the vector of parameters to be estimated: \( p_{est,j} = p_{ij}, \ i_j \in \{1, \ldots, n_p\}, \ j = 1, \ldots, n_{p_{est}}, \ n_{p_{est}} = 116 \). Our software infrastructure is benchmarked for the following tasks:

1. Simulate the original model
2. Compute value of the objective function \( \Phi \).
3. Compute the gradient \( \partial \Phi / \partial p_{est} \) by means of first-order adjoint sensitivity analysis.
4. Compute the gradient \( \partial \Phi / \partial p_{est} \) by means of first-order forward sensitivity analysis.
5. Compute the Hessian \( \partial^2 \Phi / \partial p_{est}^2 \) by means of second-order adjoint sensitivity analysis.

**Code Generation and Compilation**

All computations are performed on a Notebook with a 2.53 MHz Intel Core2 SP9600 processor, equipped with 4 GB RAM and running Linux Mint 12.

As illustrated in Figure 4, the first task of the JADE infrastructure is to generate C-code from a flat Modelica model. This done by the Mod2C compiler, which generates a C-function of the model residual and related utility functions, e.g., for providing access to the variable names. This part of the code generation takes roughly 4 seconds. Then, the derivative code compiler ddc, an algorithmic differentiation (AD) tool, is applied for generating derivatives of the model residual. This part takes approximately 5 minutes, thereof 4 minutes for the generation of the second-order adjoints of the model residuals.

The generated code, including the derivative codes, is then compiled either in a dynamic link library (DLL) on Windows platforms or a shared object on Linux or UNIX platforms. Here, the compilation times strongly depend on the compiler flags, especially on the optimization flags. The sequential compilation times with the g++-4.6.1 compiler of the GNU Compiler Collection (gcc) are 2 minutes (thereof 1 minute for the second-order adjoints) for non-optimized code, and for optimized code (-O3-flag) 53 minutes (thereof 37 minutes for the second-order adjoints).

**Simulation and Sensitivity Analysis**

We apply the JADE infrastructure for simulating and evaluating the objective function, as well as its gradient and Hessian with either optimized or non-optimized compiled code. The numerical kernel relies on the NIXE integrator. NIXE implements the extrapolated linearly-implicit Euler method, and provides facilities for higher-order forward or adjoint sensitivity analysis. In detail, NIXE implements a modified discrete adjoint method for the adjoint sensitivity analysis [5]. Further, since the objective function \( \phi \) in Equation 8 depends on different points in time, we use the technique of composite adjoints [4], instead of the classical adjoint sensitivity analysis (which only submits one final time) [1]. Whenever the gradient or Hessian of a DAE-embedded functional of the type \( \phi(x(t_1, p), \ldots, x(t_N, p)) \) with respect to sufficient many parameters has to be computed (cf. Equation 8), from the view of computational efficiency, composite adjoints are the method of choice. Roughly spoken, composite adjoints compute a linear combination of the \( N \) classical adjoints associated
with $\phi(x(t_1, p), \ldots, x(t_N, p))$ corresponding to the final times $t_1, \ldots, t_N$. The computational cost of composite adjoints is equivalent to the cost of only one classical adjoint computation with a final condition at $t_N$. For details we refer to [4].

Table 1 shows the performance of different computations. For comparison, we have also executed a simulation with Dymola 7.1 in combination with MS Visual Studio 2008 on the same notebook but running Windows 7 (see last row of Table 1).

<table>
<thead>
<tr>
<th>Task</th>
<th>JADE results, Abs Tol=$10^{-5}$</th>
<th>1488 state variables, 116 parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time</td>
<td>Optimized</td>
<td>Non-opt.</td>
</tr>
<tr>
<td>Simulation</td>
<td>1.6 s (DASSL, Tol=$10^{-3}$)</td>
<td></td>
</tr>
<tr>
<td>Objective</td>
<td>10.5 s</td>
<td>14.5 s</td>
</tr>
<tr>
<td>Gradient (adjoint)</td>
<td>14.5 s</td>
<td>19.9 s</td>
</tr>
<tr>
<td>Gradient (forward)</td>
<td>46.8 s</td>
<td>63.5 s</td>
</tr>
<tr>
<td>Hessian (2nd adjoints)</td>
<td>180.0 s</td>
<td>465.0 s</td>
</tr>
<tr>
<td>Dymola Simulation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The simulation time of JADE is competitive with Dymola, for both the optimized and the non-optimized compiled codes. However, Dymola does neither support first-order nor second-order sensitivity analysis. We observe that the evaluation the objective function takes much longer than the simulation. This is due to the NIXE integrator stopping at the measurement times and resetting the adaptive step size control.

Computing the $116$ components of the gradient with adjoint sensitivity analysis takes only about $1.5$ times the time of one single function evaluation for both the optimized and the non-optimized codes. Forward sensitivity analysis is $3$ times slower (optimized and non-optimized). Computing the $116 \times 116$-Hessian matrix takes $180$ seconds with the optimized compiled code and $465$ seconds with the non-optimized compiled code.

**Comparison with Finite Differences**

If we compare the computational times of the JADE sensitivity analysis with the costs of finite differences, we clearly see the superiority of the tailored numerical methods of JADE. Table 2 shows compute time ratios of the different sensitivity tasks as compared to a single objective function evaluation.

The cheapest finite differences formulas would require $117 = 1 + 116$ function evaluations for the gradient and $6845 = 1 + 116 + 116^2/2$ function evaluations for the Hessian. The excellent numerical performance of the JADE prototype is mainly achieved by combining the AD tool dcc [7] with composite adjoints [4] that are computed with the specifically tailored numerical integrator NIXE [5], which strongly exploits the structure of the underlying (adjoint) sensitivity equations.

**Conclusions**

We have introduced the JADE platform for first- and second-order sensitivity analysis of DAE models. The platform combines code generation, algorithmic differentiation and a customized numerical integrator for forward and adjoint sensitivity analysis. The presented results in particular for computing the Hessian of the studied parameter estimation objective function are more than competitive. The complete $116 \times 116$ Hessian of the objective function is computed at the cost of $18$ single function evaluations, yielding accurate second-order derivatives. In comparison, computing the same Hessian with the cheapest and least accurate finite difference formula would require $6845$ function evaluations. This makes the JADE platform particularly attractive for large-scale applications with nonlinear numerical optimization solvers that require second-order derivatives.

**Outlook**

Up to now, the numerical methods of JADE are restricted to smooth Modelica models without discontinuities. However, many systems, e.g., from engineering or biotechnology, need to be modeled with non-smooth differential-algebraic equations. In addition, the modeling process can yield under-determined differential-algebraic systems (more variables than equations). In this case, some of the model variables must be determined by external criteria, for example by means of an optimization criterion. The resulting models do not belong to the well-known class of hy-
brid DAE systems, but a novel class of **non-smooth DAE-O** systems can be defined, where the “O” denotes optimization. The concept of the JADE prototype, i.e. combining a high-level model language like Modelica with algorithmic differentiation and tailored numerical solution methods, will be extended to the classes of non-smooth DAE and DAEO systems.

**References**


