

SBML_odeSolver: a simple libSBML and CVODE based ODE Solver for numerical analysis of biological reaction networks

Rainer Machne^a, Christoph Flamm^a

^aInstitut für Theoretische Chemie und Molekulare
Strukturbiologie, Universität Wien, Währingerstraße 17,
A-1090 Wien, Austria

Contents

1	SBML and CNode based ODE Solver	1
1.1	Summary	1
1.2	Introduction	2
1.3	Usage and Basic Architecture	4
1.3.1	Constructing ODEs from Reaction Networks	5
1.3.2	Integrating ODEs Numerically	8
1.3.3	Visualizing Structure and Dynamics	9
1.4	Integrated Result Visualization	12
1.5	Accuracy and Testing	13
1.6	Outlook	14

1 SBML and CCode based ODE Solver

1.1 Summary

Abstract The SBML_odeSolver is a simple command-line tool for (1) constructing a system of ordinary differential equations (ODE) from chemical reaction networks and (2) numerically integrating the time course of concentrations of chemical species and (3) basic visualization of model structure and integration results. It is based on SBML, the recently developed standard for description of biological reaction networks, the SBML library libSBML for parsing SBML and constructing the ODE system, and on CCode for numerical integration of the derived system of ODEs. Optional data visualization modules allow printing of integration results directly to Grace and drawing graphs of the reaction network, and a Jacobian interaction graph of the ODE system via graphviz' graph drawing library.

Availability <http://www.tbi.univie.ac.at/~raim/odeSolver>

1.2 Introduction

Background Mathematical modeling of chemical reactions, and especially biochemical reaction networks involves a variety of techniques and theories and has long been applied for various purposes in research and technology. Diverse but potentially complementary approaches have been taken to analyze networks of chemical reactions, roughly dividable in ‘dynamical’ and ‘structural’ analysis.

Dynamical analysis tries to understand the time-dependent development of reaction rates and molecular concentrations, including intuitively hardly recognizable properties that ‘emerge’ due to complex feedback cycles within reaction networks. Given a complete reaction network, including a rate law description for each reaction, one can either derive a system of ordinary differential equations (ODE) for the time-change of the participating chemical species, or a so-called chemical master-equation for discrete stochastic modeling. Both formulations assume a well-stirred homogeneous solution of all reactants. If interested in diffusion regulated processes the researcher can set up a series of partial differential equations (PDE), additionally describing space-dependence of the concentration of chemical species. Several other approaches adapted from various mathematical and computational techniques have been explored, including multiple agent systems, petri nets [9, 4], which naturally resemble a bipartite reaction graph and its stoichiometry, or the π -calculus for analysis of concurrent parallel processes [11], and grammar models, semantical and logic descriptions.

Some of the latter methods overlap with the second class, the ‘structural’ network analysis. Those methods include graph theory based approaches to describe global network structure, that are essentially ‘graph walking’ and ‘graph partitioning’ problems. More specialized techniques from theoretical chemistry, such as mass conservation analysis, metabolic control or regulation analysis, allow to identify e.g. sensitivities of the reaction network to a subset of parameters or identify separable modules such as so-called ‘elementary flux modes’. Another interesting class of computational models of reaction network would be constituted by the already wide range of metabolic/regulatory pathway databases.

SBML - the Systems Biology Markup Language Accordingly, many tools for all kinds of computing platforms have been created, each relying on their own format for describing reactions, initial conditions, parameters

and rate laws. The need for exchange and merging of models motivated collaborative efforts to develop a standard format for describing the common chemical reaction networks underlying the various derived mathematical descriptions. Of two competing XML based formats, SBML (Systems Biology Markup Language) [15, 2, 5] and CellML (Cell Markup Language) [8] the former now seems to be widely accepted in the modeling community and is supported by a growing number of long-existing as well as newly emerging tools.

Motivation The available tools (see e.g. website [15]) cover a variety of methods to edit and analyze a reaction network and its dynamics and/or structure. However, they are mostly designed as platform specific standalone tools whose functionality is only accessible via more or less complex user interfaces.

The SBML_odeSolver in its first released version (1.0) is a minimal ODE construction and integration tool with some additional (optional) features for graph drawing and result visualization, entirely written in C and based mainly on libSBML, the C/C++ library for parsing and editing SBML [7], and CVode, a stiff and non-stiff ODE solver in C [12], the same tool that is also used in SCAMP, a classic tool for model simulation and metabolic control analysis [13]. The SBML_odeSolver is targeted at bioinformaticists, biomathematicians and ‘command-line friendly’ biochemists and biologists.

Possible Applications Through its easy-to-use and stripped down functionality, the SBML_odeSolvers offers itself for a variety of purposes, both as a stand-alone tool for quickly exploring system structure and dynamics and as a simple and reliable module surrounded by other additional and higher-level analysis or visualization tools. The program might be most interesting for a use in batch integration of models, e.g. via a calling script or program that interprets results and changes SBML structure or parameters accordingly. Such a use is indicated by the green path in figure 1. Examples for a possible usage of the program via short Perl scripts, depending on the Perl5 binding for libSBML, are included in the distribution.

Our own working group is interested in the **evolution of network dynamics**, and we will use the tool to quickly identify if a given gene regulatory network, evolved by mutations, fulfills specific dynamics such as oscillations or multiple steady states (multi-stationarity). Another obvious use would be

to test if a parameter set, derived from an optimization technique, leads to the desired dynamics as e.g. measured in experiments. **Parameter optimization** and the **inverse problem of chemical kinetics** would be the buzzwords for this area of research.

At this point it is worth pointing out that the SBML_odeSolver's use is not restricted to chemical or biological problem. Through libSBML's formula parsing and data structure, the SBML_odeSolver opens CCode for a use with general ODE systems. SBML can encode any system of ODEs, although the species wouldn't correspond to chemical species, against SBML definitions. In fact the program itself produces such an actually invalid model to represent ODE systems (see chapter 1.3.1). Thus, for the cost of ignoring strict SBML definitions, one could use the program as a **general ODE solver**.

Last but not least it should be emphasized that the SBML_odeSolver's development has always considered its potential as a convenient tool for **educational purposes**. The program might comprise a nice tool to introduce the principals of chemical reaction networks and the standard SBML, reproducible ODE construction from such reaction networks and the use of libSBML for handling reaction networks computationally to students. Plans for further development of the tool will especially consider easy and quick, and informative visualization of model structure and dynamics.

1.3 Usage and Basic Architecture

The SBML_odeSolver-1.0 is a very simple, command-line driven ANSI C program, stripped down to the basic functionalities of

- (1) **construction of an ODE system**
from an SBML encoded reaction network
- (2) **numerical integration of an ODE system**
encoded in a defined subset of (semantically incorrect) SBML
and
- (3) **printing and basic visualization**
of model structure and integration results

It is distributed as source code and can be compiled via the usual configure/make procedure requiring the automake tool to be installed. See the file INSTALL in the distribution for detailed instructions.

Table 1 lists all available procedures and the command-line options to call them. Figure 1 shows the program's work-flow of data parsing, data conversion, ODE construction, ODE integration, and output of the program. The steps (1-3) are labeled as above. Plain-text nodes represent accessible data, while elliptic nodes represent program functionality. The green path indicates a possible use by an external script or program. Each step is described in detail in the following chapters 1.3.1-1.3.3. For more details, please consult the extensive documentation of the source code.

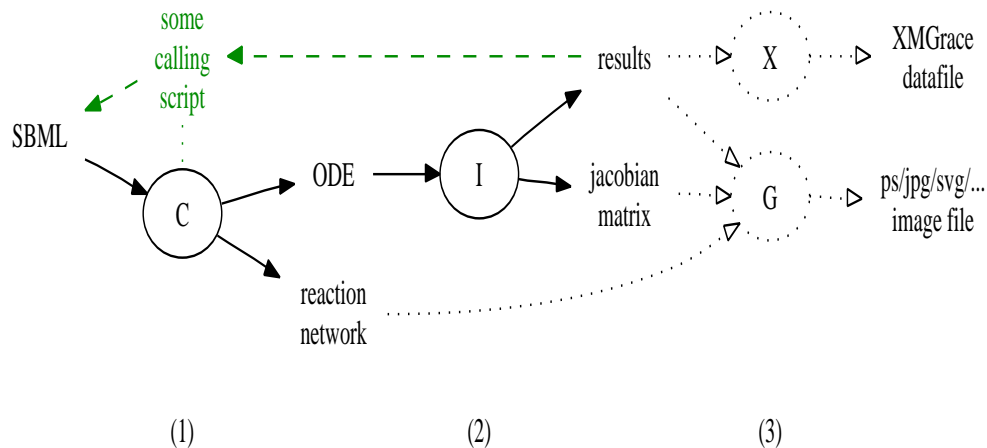


Figure 1: Basic data-flow architecture of the SBML_odeSolver, see chapters 1.3.1-1.3.3 for details

1.3.1 Constructing ODEs from Reaction Networks

See node **C** in figure 1: SBML parsing, ODE construction, data conversions; depending on libSBML.

The simple SBML_odeSolver makes heavy use of the ANSI C/C++ SBML library libSBML [7] for parsing SBML encoded reaction networks and constructing ODEs and other formulas and finally for evaluating their current values, e.g. during an integration run. The libSBML's Abstract Syntax Tree (AST) convention for representation of mathematical formulas was especially

useful for the latter purpose.

The steps implemented by the functions subsumed in node **C** of figure 1 can be outlined as follows (*italic* symbols *ODE.xml* and *SBML.xml* resemble nodes ODE and SBML in the figure):

- **C.1 Load, validate and parse SBML file**

The input file is an SBML encoded model *SBML.xml* of chemical reactions and all other possible SBML definitions. LibSBML provided functions are used to parse the model, and access its data in the following steps. The data can optionally be validated towards SBML's schema definitions before anything else is done.

- **C.2 Copy predefined ODEs**

A new model *ODE.xml* with compartment size 1 is created; predefined ODEs, i.e. SBML 'rate rules' are copied from *SBML.xml*, and their variables added as species, to the new model. Note that, therefore in this new model *ODE.xml* not all 'species' do necessarily represent chemical compounds. Parameter values or the compartment sizes can be described by a 'rate rule' in *SBML.xml*. Thus, in *ODE.xml*, the units are ignored, the compartment field is meaningless (always set to size 1) and thus, the constructed SBML is semantically incorrect SBML. It is valid, however, against the schema definitions!

- **C.3 Construct ODEs from reactions**

For all yet undefined species, that have their 'boundaryCondition' and 'constant' fields set to 'false', an ODE is constructed from all reactions that consume or produce the species, i.e. where it appears in the list of reactants or products of the reaction definition. The ODE is constructed directly as a libSBML AST, combining SBML's 'kinetic law', 'stoichiometry' or 'stoichiometry math' definitions and the species' compartment size, or id for non-constant compartments. Local parameters, definable for 'kinetic laws' are replaced in the formulas, i.e. their name is replaced by their value.

As an example consider the two reactions in a homogeneous and continuously stirred compartment of size V :



The resulting ODE for the concentration of C, denoted [C], would add up from the two reactions' kinetic laws each multiplied with the species' stoichiometry and set positive for producing or negative for consuming reactions:

$$d[C]/dt = (+ 1 * K1 - 2 * K2) / V$$

Please consult basic text books like [1] for the details on constructing ODEs from reactions. One notable difference between the usual process and SBML specific ODE construction lies in SBML's 'kinetic law' formula that differs from the usual rate law in that its units are amount/time instead of concentration/time. This facilitates ODE construction from multi-compartmental models and, according to the SBML Level 2 Version 1 specifications [2], only requires the division of the resulting ODE by the compartment volume to obtain the usual concentration/time description. The new ODE's AST is added as a 'rate rule', i.e. an ODE describing the concentration of a species, and the corresponding species to *ODE.xml*. The species' compartment is the default compartment and its initial values are set to initial concentration.

The new model *ODE.xml* now constitutes a usual 'initial condition problem', it consists of ODEs and the initial values of their variables.

- **C.4 Copy incompatible SBML structures**

SBML's 'algebraic rules', that are needed in systems of differential algebraic equations (DAE) cannot be interpreted in terms of ODEs, and neither can discrete 'events'. Such structures are just copied to the new model, for print-out and analysis with other tools.

- **C.5 Replace constants, assignments and defined functions**

User defined functions, assigned variables and constant parameters, species and compartment of *SBML.xml* are replaced in all ODEs ('rate rules') and the copied incompatible structures (from step C.4) of the new model *ODE*.

At this point the contents of the input SBML model as well as the derived ODE system can be printed out to inspect reactions, initial conditions and equations. The new model can be printed as SBML, and this way the program can essentially be used as a conversion tool, condensing an SBML encoded reaction network to an ODE system, encoded in a defined small subset of SBML, including potentially incorrect SBML semantics.

1.3.2 Integrating ODEs Numerically

See node **I** in figure 1: Jacobian matrix construction, ODE integration; depending on CCode and libSBML.

LibSBML's abstract syntax tree (AST) represents formulas in their correct precedence encoded in tree structure. A simple recursive function, that is also included as an example program in the libSBML distribution, is used to evaluate AST formulas in the functions described below.

The simplified SBML model *ODE.xml* is used to fill an internal data structure used by the integrator function. The Jacobian matrix of the ODE system is generated in symbolic form, again as an AST. Note that, at the moment, in the exact procedure of the program, this functions takes the old model *SBML.xml* and calls the above described function to obtain *ODE*.

An integrator function then initializes and calls CCode, an ANSI C tool for solving non-stiff and stiff ODE systems [12], and provides CCode with a function that evaluates the AST representation of the ODEs and (optionally) the Jacobian matrix of the ODE system with current values (current species concentrations), whenever this is requested by CCode's integration method. The integration methods employed by CCode are variable-coefficient forms of the Adams and BDF (Backward Differentiation Formula) methods, and simple functional (or fixed point) iteration or a variant of Newton iteration for non-stiff and stiff problems respectively. Please consult CCode's user guide [12] for more detailed information about method and implementation. The SBML_odeSolver uses the BDF method and Newton iteration with the CCode dense linear solver which can solve both stiff and non-stiff systems. The integrator function has been derived from CCode's example program 'cvdx.c'. It requires the current values of the Jacobian matrix. These can either be calculated from their AST representations or by CCode's internal

approximation of the Jacobian. The latter occurs if (a) the ODEs include expressions, whose differentiation is currently not implemented, (b) the solver produces errors with the generated Jacobian but not with the internal approximations (the reasons of which have yet to be determined in detail) or if (c) the user chooses so via command-line options. CCode uses absolute and relative error tolerances for each calculated time step. The absolute and relative error tolerances are set to 10^{-18} and 10^{-14} , respectively, and can be set via a command-line option. The accuracy required by published tests (see 1.5) could be achieved easily by setting the absolute error in the range of 10^{-21} to 10^{-18} . For some problems the user will also have to adjust the maximum number of steps that CCode tries to reach the next requested time step within the error tolerances. Table 1 lists all available command-line options. If CCode integration fails an error message is printed. The given error flags are explained in table 4. In any the final output of the CCode module is a set of statistics. e.g. how many internal steps, how many calls to ODE or Jacobian evaluation were needed. Please consult table 3 for interpretation of this output.

Discrete Events SBML allows to specify discrete events, in which a variable's value triggers the resetting of other variables. Such discrete events can lead to discontinuous and are not defined in the realm of ODE systems. The SBML_odeSolver in version 1.0 implements a provisional event evaluation which can be activated via a command-line option (see 1). At each printed time step, the event triggers are evaluated. Upon triggering of an event, the integrator stops and is restarted with new values. This event detection is not exact. The accuracy of event detection depends completely on the chosen print-step interval!

1.3.3 Visualizing Structure and Dynamics

The odeSolver prints all data to stdout, and messages to stderr, as a default. The data should then be processed by other tools. However, it also offers some additional functionalities for quick and easy exploration of structure and dynamics of a reaction network model. Via command-line options the program can be used to print model contents instead of integrating. Two optional modules that depend on additional libraries are used to support visual exploration of the model. In the interactive mode the user has some additional possibilities for processing of data.

Interactive Mode Via an interactive mode the user has access to most functions that are available via command-line options. The user can inspect a loaded SBML model, construct and view the ODEs, integrate them, store and view integration results. Additionally the interactive mode allows to set alternative initial conditions and print phase diagrams for two species to XMGrace. The interactive mode is especially helpful when exploring a new SBML file with a structure unknown to the user and in the lack of other tools. It might especially find appreciation for educational purposes as outlined above.

Result Visualization using XMGrace See node **X** in figure 1: result visualization with XMGrace; depending on the grace library `np_grace`.

Instead of printing integration results to a file the user can choose to directly visualize concentration/time graphs in XMGrace [20]. See Table 1 for other output data. The interactive mode additionally allows to select 2 species to draw 2-dimensional phase diagrams to XMGrace (see lower images in figure 2).

Graph Drawing using graphviz See node **G** in figure 1: reaction network and Jacobian matrix graph drawing with graphviz; depending on the graphviz library.

The reaction network can be drawn as a bipartite graph of molecules and reactions, based on graphviz' algorithms for graph layout (graph drawing, graph embedding) [3]. Edges from chemical species to and from reactions are labeled with the corresponding stoichiometry. The generated graphic files can easily be used for exploration of the structure of the reaction network. A species interaction graph based on the non-zero entries of the Jacobian matrix can be constructed via graphviz. Edge colors and labels are set by the value of the corresponding entry in the Jacobian matrix at some chosen time point of integration. Negative influence of a species on the ODE of another species is represented by a red arrow, positive influence by a black arrow. The exact values are the labels of this graph. This graph is well suited for visually exploring the dynamic regulation of the network, e.g. to get a first impression on possible and relevant positive or negative feedback loops within a reaction network. The upper left image in figure 2 shows such a

graph for the MAPK pathway’s phosphorylation cascade with a theoretical negative feedback. The upper right image is the reaction network of the same model. This model by Kholodenko et.al.[6] has been obtained from the official SBML model repository at <http://www.sbml.org/models>.

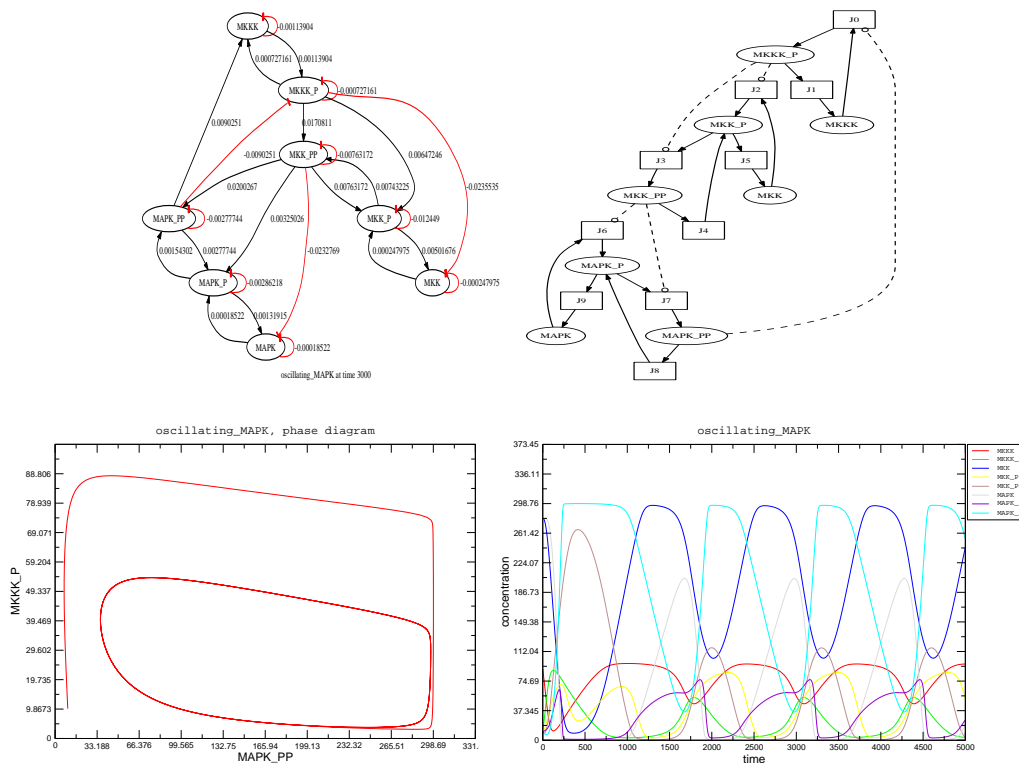


Figure 2: Example results for a model by Kholodenko et.al. 2000 [6], (taken from <http://www.sbml.org/models>)

Compilation without optional modules Both functionalities, grace- and graphviz- dependent, are optional. The configure script recognizes if the necessary libraries are available and, if not, the program can be compiled without these functions. Compilation without the Grace library will cause the program to just ignore the command-line option and print out results as text, while compilation without graphviz will lead to printout of graphs as text files without calculated coordinates in the graphviz’ ‘dot’ format.

1.4 Integrated Result Visualization

The Perl wrapper script ‘bioLog_resultVisualizer’ (*rV*) exemplifies a very simple use of the program for both direct and higher-level visualization of simulation results. The script uses SBML_odeSolver’s and Perl’s graphviz modules to generate SVG based graph drawings and embeds them in a set of crosslinked html files. The SVG files (chemical species, reactions) are animated by the results of a simulation run and link to sites with detailed model and result information for each species and reaction.

BioLogic Result Interpretation Additionally the script searches for two other, already existing files, which can be created to embed the SBML model and display the results of an animation within some higher-level, hand-written, representation of the reaction network model.

A hand written *indexfile*, that is parsed by the *rV* script, lists all proteins or otherwise defined higher-level (modular) entities in the model system, and each protein/module is accompanied by a list of chemical species in the SBML model that represents different states of the protein (e.g. chemically modified or bound within a complex). For each entity there must be at least one chemical species, that is marked as ‘active’.

The second file is an SVG based diagrammatic representation of interactions between the above defined modular entities, similar - in the example models - to the well-known diagrams in cell-biological and medical literature describing cellular regulation processes and experiments. This *bioLog* activation/inhibition diagram is encoded again as a graph file and graphviz was used create the SVG images of the graph drawing. The modular entity list (proteins or defined processes in the example models) and its activity tags are used to process simulation results to active/non-active ratios and visualize this time series within the graph SVG file.

The example shown in figure 3 is a *bioLog* activation/inhibition shema of a published model of receptor mediated activation of the so-called ‘Mitogen Activated Protein Kinase’ [16], a eukaryotic module of cellular signal transduction pathways used as a ‘switch’ or ‘amplifier’ of externals and internal signals in diverse contexts of cell regulation, like growth, cell-cycle, differentiation, migration, adhesion and apoptosis. The simulation of this model is based on an SBML model initially obtained from SigPath and adapted by hand. The *indexfile* and the *bioLog* diagram are hand-written. An animated and hyperlinked set of result files for this model can be browsed at

http://www.tbi.univie.ac.at/~raim/schoeberl_02/index.html.

The *rV* wrapper script is written in PERL5 and dependent on Perl modules SVG::Parser, GraphViz, GD::Graph, and the newly developed LibSBML bindings for the SBML library libSBML.

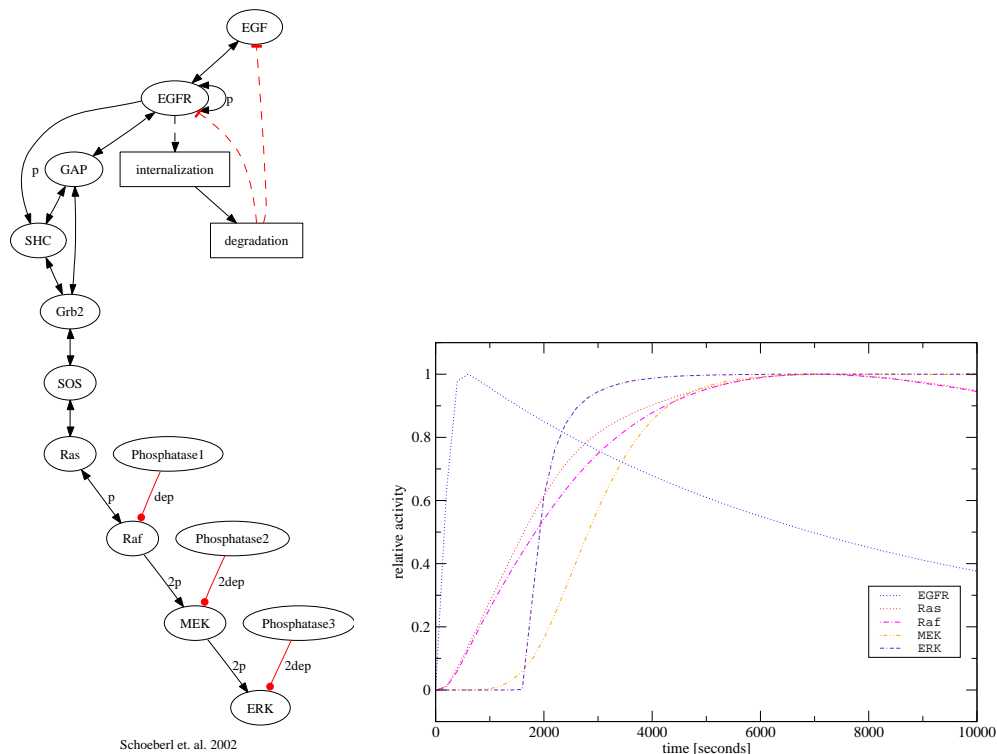


Figure 3: Example bioLog schema of Schoeberl et. al. 2002 [16]

1.5 Accuracy and Testing

The SBML_odeSolver has been extensively tested with the first published version of the ‘SBML Semantic Test Suite’ provided by the SBML team (see website [15]). All of the SBML test models that do not include

- (a) algebraic rules, which can only be solved with methods of DAEs (Differential Algebraic Equations),
- (b) events, that would require approximations of the exact event time and

(c) delays, that would require methodology of solving ‘ODEs with delays’ could be successfully integrated, except for one including a cube root expression for which at the last given time point the SBML_odeSolver’s result deviates from the target results produced with MathSBML, a SBML package for Mathematica [17]. The test suite includes models at the extremes of low numerical values, and they were solved without problem. However, models handling very low amounts, in the range of circa 0 to 1000 molecules (per cell), a stochastic solver will be required.

Detailed results of this test run, and instruction how to reproduce the tests are distributed with the source code.

1.6 Outlook

At moment the tool can process all SBML Level 1 + 2 definitions and numerically integrate the dynamics of models that are interpretable within the realm of ordinary differential equation systems and thus numerically solvable with CVode. Some possible further developments, through additional internal functionality or integration with other tools, are outlined in the following. The maintenance of the tool will include a more detailed use of CVode and its various methods for integration as well as refinement of internal batch integration, result printing and processing, and the testing of use and communication with other programs.

DAEs, Hybrid models, PDEs Discrete ‘events’ and ‘delays’ cannot be interpreted by CVode. The detection of discrete events - or at least a useful approximation thereof, using batch integration with CVode - is, however planned as an internal extension of the presented tool. SBML models containing definitions for ‘algebraic rules’ call for a separate module solving systems of differential algebraic equations (DAE).

Level 3 of SBML will also come up with some definitions, including spatial models, that won’t be interpretable by the tool at this stage. Separate modules for constructing and solving PDE systems, describing e.g. morphogenic activity during development or the interpretation of chemical gradients by chemotacting cells ranks high on our interest- and todo-lists, but are each ... a lot of work.

Structural Analysis Tools for mass conservation analysis [10, 14] would allow to reduce the amounts of equations. Identification of relevant parameters and elementary flux modes by methods of metabolic regulation analysis, would help to extract interesting subsystems from large models. This can again be useful for theoretical parameter optimization approaches (see below) but also offer interesting possibilities for the (graphviz dependent) model visualization module and the result visualizer wrapper (chapter 1.4).

The information necessary for above described biologic result visualization, the *indexfile* could be automatically generated, given only the active state(s) of an entity, while all other (inactive) states of the entity should be deducible by mass conservation. Moreover, even the causal interactions of the *bioLog* interaction diagram could be automatized, if additionally ‘input’ and ‘output’ species can be defined, when interpreting the network as a module - of signal transduction in our examples. Such a higher-level embedding of a reaction network could employ graph search and partitioning algorithms to identify relevant higher-level causal relations - e.g. dominant cascades or feedback cycles - in the reaction network but also in the Jacobian matrix of the derived ODE system.

Structure and Evolution of Gene Regulatory Networks Having the ODE system and its Jacobian matrix in symbolic and interpretable form, motivates for approaches to identify and analyze positive and negative feedback cycles of a system. This could help reducing the system’s dynamics to higher-level discrete or logic models of system behavior. Such biochemical feedback cycles constitute basic biological regulation modules [18, 19] realizing both stable oscillatory behavior, e.g. in cell cycle or cell migration, or stable stationary states, leading to differentiation, cell adhesion in (epithelial) tissues or e.g. directed migration. Cell-biological and medical experimentation operates much closer to this higher-level descriptions of function than to basic reaction networks as encoded e.g. by SBML.

The Inverse Problem of Chemical Kinetics The ‘inverse problem of chemical kinetics’, i.e. parameter optimization towards desired system dynamics, as e.g. measured in experiment or conjectured in theory, would constitute an obvious application for a refinement of the internal batch integration and parameter variation functionalities. Moreover it is planned to implement an interface to external function evaluation, that will be useful

to integrate the SBML_odeSolver with sophisticated parameter optimization algorithms that are currently developed at our institute.

(Collaborative) Experiment Design Cell biological and medical knowledge of the gene regulatory and signaling reaction networks is mostly closer to above mentioned higher-level logical (*bioLogical*) models and this knowledge is often represented in activation/inhibition schemes. Such diagrams in literature are poorly defined in their node and edge meaning but interpretable representations of a specific process and the current understanding thereof. The lack of definitions is actually their power in representing the diverse mechanistics of cell-biological phenomena. However, if its possible and useful to derive such logic models from underlying reaction networks, and their feedback regulation, a top-down approach of such methods might help to extract possible network structures and relevant parameters from an experimentally known or theoretically conjectured higher-order logic model, as represented by such ‘causal graphs’ in cell biological and medical literature, and from incomplete knowledge on the exact mechanics.

Last words The SBML_odeSolver was programmed and will be maintained and extended for our own purposes in one or more of the many named directions. We hope, however, to raise some interest for the application and find users, who will be welcome to participate in further development. The program is written very close to libSBML and some of its functions might be of interest to other libSBML users.

Table 1: Usage and command-line options for the SBML_odeSolver

USAGE: odeSolver <sbmlfile.xml> [OPTION(s)]

Options	Argument	Description
GENERAL OPTIONS		
-h	-help	Print usage information
-i	-interactive	Start the interactive mode
	-gvformat string	Output format for graphviz module
SBML FILE PARSING		
-v	-validate	Validate SBML file
	-model string	SBML file name 'sbmlfile.xml'
	-mpath path	Set Model file path
	-spath path	Set Schema file path (default: mpath)
(1) PRINT REACTIONS AND DERIVED ODEs		
-e	-equations	Print model and derived ODE system
-o	-printsbml	Construct ODEs and print as SBML
-g	-modelgraph	Draw graph of reaction network

continued on next page ...

Table 2: Usage and command-line options for the SBML_odeSolver, continued

... continued from previous page

(2) INTEGRATION PARAMETERS

-f	-onthe-fly		Print results during integration
-j	-jacobian		Toggle use of the jacobian matrix
-s	-steadyState		Abort integration at steady state
-n	-event		Detect and evaluate events (do not abort). ACCURACY DEPENDS ON STEP SIZE!!
	-param	string	Choose parameter for batch integration, from 0 to value in 50 steps
	-printstep	integer	Time steps of output (default: 10^3)
	-time	float	Integration end time (default: 10^3)
	-error	float	Absolute error tolerance (default: 10^{-18})
	-rerror	float	Relative error tolerance (default: 10^{-14})
	-mxstep	integer	Maximum step number (default: 10^5)

(3) INTEGRATION RESULTS

-a	-all		Print all available results
-y	-jacobianTime		Print time course of Jacobian matrix
-k	-reactions		Print time course of the reactions
-r	-rates		Print time course of the ODEs
-w	-write		Write results to file or save XMGrace file
-x	-xmgrace		Print results to XMGrace
-m	-matrixgraph		Draw Jacobian matrix graph

Table 3: Final output: CVode integration parameters and statistics

Short Meaning**CVode integration parameters:**

mxstep	maximum number of steps CVode used at each internal time step $ h $
rel.err	relative error tolerance at each internal time step $ h $
abs.err.	absolute error tolerance at each internal time step $ h $

CVode integration statistics:

nst	cumulative number of internal steps taken by the solver
nfe	number of calls to the ODE evaluation function ‘f’
nsetups	number of calls to the linear solver’s setup routine
nje	number of Jacobian evaluations, i.e. either calls to the function that evaluates the automatically generated Jacobian matrix expressions or the internal approximation CVDenseDQJac.
nmi	number of NEWTON iterations performed.
ncfn	number of nonlinear convergence failures that have occurred
netf	number of local error test failures that have occurred

Table 4: CNode failure messages

Flag	Message	Description
0	SUCCESS	CNode completed integration.
-1	CVODE_NO_MEM	The cnode_mem argument passed to CNode was null. This error should not appear in the SBML_odeSolver.
-2	ILL_INPUT	One of the inputs to CNode was illegal, including the situation when one of the error vectors becomes ≤ 0 during CNode's internal time stepping. The printed error message will give specific information. In the SBML_odeSolver, this failure occurs when e.g. the out-time passed was '0'.
-3	TOO_MUCH_WORK	The solver took a maximum of internal steps but could not reach the next print-step. The default step number is 100000; it can be set with command-line option '-mxstep'.
-4	TOO_MUCH_ACC	The solver could not satisfy the accuracy demanded (via options -error and -rerror) for an internal time step.
-5	ERR_FAILURE	Error test failures occurred too many times during one internal time step or occurred with $ h = h_{min}$, i.e. the necessary internal time step became too small.

continued on next page ...

Table 5: CNode failure messages, continued

Flag	Message	Description
... continued from previous page		
-6	CONV_FAILURE	Convergence test failures occurred too many times during one internal time step or occurred with $ h = h_{min}$. This can sometimes happen either with or without using the automatically generated Jacobian matrix. That is why the the SBML_odeSolver tries to integrate again upon this error, but now without or with use of the Jacobian matrix (resetting option '-j'). In other cases this error can be avoided by allowing a bigger error tolerances
-7	SETUP_FAILURE	The linear solver's setup routine failed in an unrecoverable manner. This error has not occurred (during test runs).
-8	SOLVE_FAILURE	The linear solver's solve routine failed in an unrecoverable manner. This error has not occurred (during test runs).

List of Tables

1	Usage and command-line options for the SBMLodeSolver . .	17
2	Usage and command-line options for the SBMLodeSolver, continued	18
3	Final output: CNode integration parameters and statistics . .	19
4	CNode failure messages	20
5	CNode failure messages, continued	21

List of Figures

1	Basic data-flow architecture of the SBMLodeSolver, see chap- ters 1.3.1-1.3.3 for details	5
2	Example results for a model by Kholodenko et.al. 2000 [6], (taken from http://www.sbml.org/models)	11
3	Example bioLog schema of Schoeberl et. al. 2002 [16]	13

References

- [1] Voit E.O. *Computational Analysis of Biochemical Systems*. Cambridge University Press, 2000.
- [2] A. Finney and M. Hucks. Systems biology markup language: Level 2 and beyond. *Biochem Soc Trans*, 31(Pt 6):1472–1473, Dec 2003.
- [3] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, Sep 2000.
- [4] R. Hofstaedt. Petri nets and the simulation of metabolic networks. *In Silico Biol*, 3(3):321–322, 2003.
- [5] M. Hucks, A. Finney, HM. Sauro, H. Bolouri, JC. Doyle, H. Kitano, AP. Arkin, BJ. Bornstein, D. Bray, A. Cornish-Bowden, AA. Cuel-
lar, S. Dronov, ED. Gilles, M. Ginkel, V. Gor, II. Goryanin, WJ.
Hedley, TC. Hodgman, JH. Hofmeyr, PJ. Hunter, NS. Juty, JL. Kas-
berger, A. Kremling, U. Kummer, N. Le Novre, LM. Loew, D. Lu-
cio, P. Mendes, E. Minch, ED. Mjolsness, Y. Nakayama, MR. Nelson,

- PF. Nielsen, T. Sakurada, JC. Schaff, BE. Shapiro, TS. Shimizu, HD. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, and . . The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, Mar 2003.
- [6] BN. Kholodenko. Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. *Eur J Biochem*, 267(6):1583–1588, Mar 2000.
- [7] libsbml.
<http://sbml.org/software/libsbml/>.
- [8] CM. Lloyd, MD. Halstead, and PF. Nielsen. Cellml: its future, present and past. *Prog Biophys Mol Biol*, 85(2-3):433–450, Jun-Jul 2004.
- [9] VN. Reddy, ML. Mavrovouniotis, and MN. Liebman. Petri net representations in metabolic pathways. *Proc Int Conf Intell Syst Mol Biol*, 1:328–336, 1993.
- [10] C. Reder. Metabolic control theory: a structural approach. *J Theor Biol*, 135(2):175–201, Nov 1988.
- [11] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. *Pac Symp Biocomput*, pages 459–470, 2001.
- [12] Cohen S. and Hindmarsh A. Ccode, a stiff/nonstiff ode solver in c. *Computers in Physics*, 10(2):138–143, March-April 1996.
- [13] H. M. Sauro. Scamp: a general-purpose simulator and metabolic control analysis program. *Comput Appl Biosci*, 9(4):441–50, August 1993.
- [14] HM. Sauro and B. Ingalls. Conservation analysis in biochemical networks: computational issues for software writers. *Biophys Chem*, 109(1):1–15, Apr 2004.
- [15] Sbml.
<http://sbml.org>.

-
- [16] B. Schoeberl, C. Eichler-Jonsson, ED. Gilles, and G. Mller. Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized egf receptors. *Nat Biotechnol*, 20(4):370–375, Apr 2002.
 - [17] BE. Shapiro, M. Hucka, A. Finney, and J. Doyle. Mathsbnl: a package for manipulating sbml-based biological models. *Bioinformatics*, 20(16):2829–2831, Nov 2004.
 - [18] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory. i. structural conditions of multistationarity and other nontrivial behavior. *Chaos*, 11(1):170–179, Mar 2001.
 - [19] R. Thomas and M. Kaufman. Multistationarity, the basis of cell differentiation and memory. ii. logical analysis of regulatory networks in terms of feedback circuits. *Chaos*, 11(1):180–195, Mar 2001.
 - [20] Xmgrace.
<http://plasma-gate.weizmann.ac.il/Grace/>.