The SBML ODE Solver Library
A Tutorial With Examples in C and Perl
Part I: Structures and Interfaces

Stefan Müller, James Lu
Johann-Radon Institute for Computational and Applied Mathematics, Linz

Rainer Machné, Christoph Flamm
Institute for Theoretical Chemistry, University of Vienna

Long Beach 2007
(also see SOSlib Poster, Nr. G23)
Thanks, Richard!
Outline

3 parts, 1 hour each:

- **Part I: SOSlib Structures and Functions:**
  basic functionality
- **Part II: Parameter Identification:**
  inverse ill-posed problems and regularization
- **Part III: Inverse Dynamical Analysis:**
  mapping oscillations and multistability onto models

**Tutorial material:**
http://www.tbi.univie.ac.at/~raim/odeSolver/ICSB07
The SBML ODE Solver Library (SOSlib) is a C/C++ programming library for construction, symbolic and fast numerical analysis of a system of ordinary differential equations (ODEs) derived from a biochemical reaction network encoded in the Systems Biology Markup Language (SBML).

Its native API extends libSBML functionality and provides interfaces to all internal data structures, symbolic operations and fine-grained wrappers around numerical routines of the SUNDIALS package (SUite of Nonlinear DIfferential/ALgebraic equation Solvers).

Website: http://www.tbi.univie.ac.at/~raim/odeSolver
Webservice: http://rna.tbi.univie.ac.at/cgi-bin/SBML_odeSolver.cgi

*A native API for symbolic and fast numerical analysis of reaction networks*. Bioinformatics 2006, 22(11):1406-7
Introduction, since Version 1.6.0

- On-the-fly compilation of equations, ca. 4x speed-up
- Forward sensitivity analysis for selected parameters AND initial conditions
- Adjoint sensitivity analysis: reading experimental data and objective functions
On-the-fly compilation of equations, ca. 4x speed-up

Forward sensitivity analysis for selected parameters AND initial conditions

Adjoint sensitivity analysis: reading experimental data and objective functions
Introduction, since Version 1.6.0

- On-the-fly compilation of equations, ca. 4x speed-up
- Forward sensitivity analysis for selected parameters AND initial conditions
- Adjoint sensitivity analysis: reading experimental data and objective functions
Introduction, TODO for v1.7.0 and v2.0.0

- SBML L2V3, SBML L3
- libSBML v3.0.0
- A stoichiometryMatrix module
- Windows version as Visual C++ project (©Andrew Finney)
- Differential Algebraic Equations, SUNDIALS IDA Solver
- Exact SBML event resolution, Delays
Introduction, TODO for v1.7.0 and v2.0.0

- SBML L2V3, SBML L3
- libSBML v3.0.0
- A stoichiometryMatrix module
  - Windows version as Visual C++ project (©Andrew Finney)
  - Differential Algebraic Equations, SUNDIALS IDA Solver
  - Exact SBML event resolution, Delays
Introduction, TODO for v1.7.0 and v2.0.0

- SBML L2V3, SBML L3
- libSBML v3.0.0
- A stoichiometryMatrix module
- Windows version as Visual C++ project (©Andrew Finney)
- Differential Algebraic Equations, SUNDIALS IDA Solver
- Exact SBML event resolution, Delays
SOSlib structures and interfaces

SOSlib Interfaces:

1. **High-level** : SBML → **Results**  
   *basic CVODES settings, SOSlib error management*

2. **NOT YET**: Stoichiometry Matrix : SBML → stoichiometryMatrix → odeModel  
   *will allow all kind of structural reaction network analysis*

3. ODE Model : SBML → odeModel ↔ Jacobian / Parametric matrices  
   *interfaces to all equations*

4. Integrator : odeModel → integratorInstance  
   *direct interface to numerical routines*

5. Variable : odeModel ↔ variableIndex ↔ integratorInstance  
   *getting and setting variables during integration*

6. Adjoint solver : comparing a model with experimental data and objective functions
**SOSlib structures and interfaces**

**OBJECTIVE FUNCTION**
- Construct Adjoint ODEs
- Construct ODEs
- SBML

**EXPERIMENTAL DATA**
- ODEs
- Jacobian matrix
- Forward solver
- Sensitivity solver
- Parametric matrix
- Adjoint solver
- Sensitivities
- Sensitivity gradients
- Event detection
- Time courses
- Multiscale Modeling

**SOSlib Interfaces:**

1. **High-level**: SBML → **Results**
   - basic CVODES settings, SOSlib error management

2. **NOT YET**: Stoichiometry Matrix : SBML → stoichiometryMatrix → odeModel
   - will allow all kind of structural reaction network analysis

3. ODE Model : SBML → odeModel ↔ Jacobian / Parametric matrices
   - interfaces to all equations

4. Integrator : odeModel → integratorInstance
   - direct interface to numerical routines

5. Variable : odeModel ↔ variableIndex ↔ integratorInstance
   - getting and setting variables during integration

6. Adjoint solver : comparing a model with experimental data and objective functions
SOSlib structures and interfaces

SOSlib Interfaces:

1. **High-level**: SBML → **Results**
   - basic CVODES settings, SOSlib error management

2. **NOT YET**: Stoichiometry Matrix : SBML → **stoichiometryMatrix** → **odeModel**
   - will allow all kind of structural reaction network analysis

3. **ODE Model**: SBML → **odeModel** ↔ Jacobian / Parametric matrices
   - interfaces to all equations

4. **Integrator**: odeModel → **integratorInstance**
   - direct interface to numerical routines

5. **Variable**: odeModel ↔ **variableIndex** ↔ integratorInstance
   - getting and setting variables during integration

6. **Adjoint solver**: comparing a model with experimental data and objective functions
SOSlib structures and interfaces

**OBJECTIVE FUNCTION**
- Construct Adjoint ODEs
- SBML

**EXPERIMENTAL DATA**
- Construct ODEs

**ADJOINT**
- Adjoint solver
- Sensitivity solver
- Forward solver
- Jacobian matrix
- Sensitivities
- Sensitivities gradients
- Adjoints solver

**Multiscale Modeling**
- event detection
- time courses

**SOSlib Interfaces:**
1. **High-level**: SBML → Results
   - basic CVODES settings, SOSlib error management
2. **NOT YET**: Stoichiometry Matrix : SBML → stoichiometryMatrix → odeModel
   - will allow all kind of structural reaction network analysis
3. **ODE Model**: SBML → odeModel ↔ Jacobian / Parametric matrices
   - interfaces to all equations
4. **Integrator**: odeModel → integratorInstance
   - direct interface to numerical routines
5. **Variable**: odeModel ↔ variableIndex ↔ integratorInstance
   - getting and setting variables during integration
6. **Adjoint solver**: comparing a model with experimental data and objective functions
SOSlib structures and interfaces

SOSlib Interfaces:

1. **High-level** : SBML → **Results**
   - basic CVODES settings, SOSlib error management

2. **NOT YET**: Stoichiometry Matrix : SBML → **stoichiometryMatrix** → **odeModel**
   - will allow all kind of structural reaction network analysis

3. **ODE Model** : SBML → **odeModel** ↔ **Jacobian / Parametric matrices**
   - interfaces to all equations

4. **Integrator** : **odeModel** → **integratorInstance**
   - direct interface to numerical routines

5. **Variable** : **odeModel** ↔ **variableIndex** ↔ **integratorInstance**
   - getting and setting variables during integration

6. **Adjoint solver**: comparing a model with experimental data and objective functions
SOSlib structures and interfaces

**OBJECTIVE FUNCTION**
- Construct Adjoint ODEs
- Adjoint solver
- Sensitivity solver
- Forward solver
- Sensitivities
- time courses

**EXPERIMENTAL DATA**
- Construct ODEs
- ODEs
- Jacobian matrix

**ADJOINT**
- Parametric matrix
- Sensitivity gradients

**SOSlib Interfaces:**
1. **High-level**: SBML $\rightarrow$ **Results**
   - basic CVODES settings, SOSlib error management
2. **NOT YET**: Stoichiometry Matrix : SBML $\rightarrow$ stoichiometryMatrix $\rightarrow$ odeModel
   - will allow all kind of structural reaction network analysis
3. **ODE Model**: SBML $\rightarrow$ odeModel $\leftrightarrow$ Jacobian / Parametric matrices
   - interfaces to all equations
4. **Integrator**: odeModel $\rightarrow$ integratorInstance
   - direct interface to numerical routines
5. **Variable**: odeModel $\leftrightarrow$ variableIndex $\leftrightarrow$ integratorInstance
   - getting and setting variables during integration
6. **Adjoint solver**: comparing a model with experimental data and objective functions
SOSlib structures and interfaces

Some Conventions:

- `integratorInstance_t`
  unlike `libSBML` structures

- `IntegratorInstance_create(odeModel_t *om, cvodeSettings_t *opt)`

- `..._create functions : require the accompanying _free function`

- `..._set / _get : returns only pointer, no free required`
  some exceptions (`_getvariableIndex`), will be streamlined with next release
int i;
Species_t *species = Model_getSpeciesById(sbml, "MAPK_PP");
cvodeSettings_t *options = CvodeSettings_createWithTime(10000, 1000);
SBMLResults_t *results = Model_odeSolver(sbml, options);
if ( results != NULL ) {
    timeCourse_t *timecourse = Species_getTimeCourse(species, results);
    for ( i=0; i<TimeCourse_getNumValues(timecourse); i++ )
        printf("%g\n", TimeCourse_getValue(timecourse, i));
}
SolverError_dumpAndClearErrors();
SBMLResults_free(results); CvodeSettings_free(options);

see file SBML_odeSolver/tutorial/integrate.c
High-level interfaces, Perl example

use ODES;
use LibSBML;

my $rd = new LibSBML::SBMLReader;
my $d  = $rd->readSBML('MAPK.xml');

my $settings = ODES::CvodeSettings_create();
=settings->CvodeSettings_setTime(10000, 100);

my $results = ODES::SBML_odeSolver($d, $settings);
$results->SBMLResults_dump();

see file SBML_odeSolver/tutorial/integrate.pl
CVODE settings for (not only) high-level interfaces

- opt = CvodeSettings_createWithTime (double EndTime, int StepNumber)
  set the end time of integration and number of time steps to be printed out

- CvodeSettings_setErrors (opt, double Error, double RError, int Mxstep)
  Absolute and relative error tolerances for integration results at single time steps (!), Mxstep is the number of steps CVODE should take at maximum to reach the next time step

- CvodeSettings_setMethod (opt, int Method, int Maxorder)
  switch between 0 (BDF) and 1 (Adams-Moulton) methods of integration for stiff and non-stiff models, resp.

- CvodeSettings_setCompileFunctions (opt, 1)
  on-the-fly compilation of the right-hand-side functions, ca. 4x speed-up

- CvodeSettings_setHaltOnEvent (opt, 0)
  to activate SOSlib's inexact SBML event handling

- CvodeSettings_setSensitivity(opt, 1)
  to activate CVODES forward sensitivity analysis

- CvodeSettings_setSensMethod (opt, int i)
  set method for sensitivity analysis: 0: simultaneous 1: staggered, 2: staggered1

- CvodeSettings_setSensParams(opt, char **sensIDs, int nsens)
  set a list of SBML IDs for which sens. anal. is requested
SBMLResults and TimeCourse structures

SBMLResults contains 5 timeCourse structures:
the time points and species, reaction flux, compartment and parameter timecourses, if calculated, the latter timecourse will also contain sensitivities

Also see Model_odeSolverBatch functions, varySettings and SBMLResultsMatrix interfaces!

see file SBML_odeSolver/examples/batch_integrate.c
for ( i=0; i<SolverError_getNum(FATAL_ERROR_TYPE); i++ )
    printf(``%s
'', SolverError_getMessage(FATAL_ERROR_TYPE, i));

SOSlib Error types:

- FATAL_ERROR_TYPE: ODE Model generation and/or integration was completely unsuccessful, existing results might be wrong
- ERROR_ERROR_TYPE: time course might not be complete, existing results should be correct
- WARNING_ERROR_TYPE: warnings
- MESSAGE_ERROR_TYPE: messages

WARNING_ERROR_TYPE:
SOSlib error management needs refactoring.
It is so far not fully consistent, and not completely NULL-safe.
In our applications however we currently don’t have any problems.
odeModel_t *odemodel = ODEModel_createFromFile("MAPK.xml");
variableIndex_t *vi = ODEModel_getVariableIndex (odemodel, "MAPK_PP");
ASTNode_t *ode = ODEModel_getOde(odemodel, vi);
printf ("d%s/dt = %s\n", ODEModel_getVariableName(odemodel, vi),
    SBML_formulaToString(ode)); /*!!! ERROR !? */
ODEModel_free(odemodel); VariableIndex_free(vi);

- The odeModel can be created from SBML models, but also directly from equations!
- Also see ODEModel_constructJacobian, ODEModel_constructSensitivity and odeSense interfaces
- The variableIndex can be retrieved from the odeModel structure
- The variableIndex is also the main data interface for the integrator!

see file SBML_odeSolver/tutorial/odemodel.c
Create ODE Model directly from equations, in Perl

```perl
my $ode0 = '((150.0 * (3.8 - (p * D_cy) - (p * C_cy)) * (1.0 - (p * C_cy))) - (9.0 * C_cy))';
my $ode1 = '(3.8 - (3.0 * D_cy) - (p * D_cy) - (p * C_cy))';

# use LibSBML function to convert the formula
# into an abstract syntax tree (AST) representation
my $astA = LibSBML::parseFormula($ode0);
my $astB = LibSBML::parseFormula($ode1);
my @AST = ($astA, $astB);

my $neq = scalar @AST;
my @x = qw/C_cy D_cy p/;  # names of variables and parameters
my @x0 = (0.0, 9.0, 0.2);  # initial conditions and constants

my $om = ODES::ODEModel_createFromODEs(@AST, 2, 0, 1, @x, @x0, undef);

Function Signature:

```c
odeModel_t *ODEModel_createFromODEs(ASTNode_t **equations,
    int NumODEs, int NumAssignments, int NumConstants,
    double *initCondAndConst,
    char **namesUsedInEquations,
    Model_t *Events);
```

see files SBML_odeSolver/tutorial/integrateODEs.pl and SBML_odeSolver/examples/integrateODEs.c
The Integrator Instance!

```c
cvodeSettings_t *options = CvodeSettings_createWithTime(10000, 1000);
odeModel_t *odemodel = ODEModel_createFromFile("MAPK.xml");
variableIndex_t *vi = ODEModel_getVariableIndex (odemodel, "MAPK_PP");
inTEGRATOR_instance_t *ii = IntegratorInstance_create(odemodel, options);
printf("# Time Course for variable %s \n", ODEModel_getVariableName(odemodel, vi));
while( ! IntegratorInstance_timeCourseCompleted(ii) )
    if ( IntegratorInstance_integrateOneStep(ii ) )
        printf("%g %g\n", IntegratorInstance_getTime(ii),
               IntegratorInstance_getVariableValue(ii, vi));
    else
        break;
SolverError_dump();
ODEModel_free(odemodel); VariableIndex_free(vi); IntegratorInstance_free(ii);
```

see file SBML_odeSolver/tutorial/integratorinstance.c
The Variable Interface

the variableIndex structure, as retrieved from odeModel, can be used to:

➤ get name (SBML ID) from odeModel
➤ get and all equations (as libSBML ASTNode) from odeModel and odeSense
➤ get current values from integratorInstance
➤ set NEW VALUES in integratorInstance also DURING INTEGRATION

SOSlib will take care of discontinuities if ODE variables x(t) are set,
for other cases setting CVODES to the (slower) TSTOP mode via
CvodeSettings_setTStop(opt, 1) is currently highly recommended!

see file SBML_odeSolver/tutorial/setvariablevalue.c
The Variable Interface, Multiscale Modeling

OBJECTIVE FUNCTION

Construct Adjoint ODEs

SBML

EXPERIMENTAL DATA

Construct ODEs

ADJOINT

Adjoint solver

Parametric matrix

Sensitivity solver

ODEs

Jacobian matrix

Forward solver

sensitivity gradients

time courses

event detection

Multiscale Modeling

see ICSB07 poster B08 by Camille Stephan-Otto Attolini
The Variable Interface, Multiscale Modeling

```
CvodeSettings_setIndefinitely(options, 1); /* integrate forever! */
CvodeSettings_setTStop(options, 1); /* RECOMMENDED! */
integratorInstance_t *ii_1 = IntegratorInstance_create(odemodel, options);
integratorInstance_t *ii_2 = IntegratorInstance_create(odemodel, options);
...
while ( ... )
{
    if ( IntegratorInstance_getVariableValue(ii_1, vi_1) < THRESHOLD )
        IntegratorInstance_setVariableValue(ii_2, vi_2, 0.0);
    IntegratorInstance_integrateOneStep(ii_1);
    IntegratorInstance_integrateOneStep(ii_2);
}
```

see file SBML_odeSolver/tutorial/setvariablevalue.c
for ( newValue = START; newValue <= END; newValue += STEPZISE )
{
    IntegratorInstance_reset(ii);
    IntegratorInstance_setVariableValue(ii, vi, newValue);
    IntegratorInstance_integrate(ii);
}

The integratorInstance is fully reusable, also with changed settings, e. g. other sensitivity parameters.
IntegratorInstance_reset(ii) and IntegratorInstance_set(ii, newSettings)
also reset variables to their initial conditions of the input SBML model!

see file SBML_odeSolver/examples/ParameterScanner.c
The Variable Interface, Parameter Scans

**OBJECTIVE FUNCTION**

Construct Adjoint ODEs

**EXPERIMENTAL DATA**

Construct ODEs

**SBML**

Construct ODEs

**ADJOINT**

Adjoint solver

Sensitivity solver

Parametric matrix

ODEs

Jacobian matrix

Forward solver

Multiscale Modeling

(1) (2) (3) (4)

**Event detection**

sensitivity gradients

sensitivities

time courses

**A**

GLN -> S

GLN

S

**A1/A2**

transcription factor S

[glutamine [molecules/fl]]

1e+07

1.2e+07

1.3e+07

1.4e+07

1.5e+07

1.6e+07

9e+06

0

50

100

150

200

250

300

**A + A1/A2**

**A1/A2**

Transcription factor S

[glutamine [molecules/fl]]

1.6e+07

1.5e+07

1.4e+07

1.3e+07

1.2e+07

1.1e+07

9e+06

0

50

100

150

200

250

300

see ICSB07 poster J12 and file SBML_odeSolver/examples/bistability.c
Evaluating Your Own Equations

Interface functions to libSBML Abstract Syntax Tree ASTNode_t *:

- **double result = evaluateAST(ASTNode_t *f, cvodeData_t *data)**
  evaluates equation \( f \) with current values of the integrator!

- **ASTNode_indexAST(ASTNode_t *f, odeModel_t *odemodel)**
  enhances evaluation speed

- **ASTNode_t *fprime = differentiateAST(ASTNode_t *f, "MAPK_PP")**
  differentiates equation \( f : \frac{df}{dMAPK_PP} \)

```c
ASTNode_t *f = SBML_parseFormula("MAPK_PP + MAPK_P + MAPK’");
cvodeData_t *data = IntegratorInstance_getData(ii);

while( ! IntegratorInstance_timeCourseComplete(ii) )
{
    if ( IntegratorInstance_integrateOneStep(ii) )
        printf("total MAPK concentration at time %g: %g
’,
            IntegratorInstance_getTime(ii), evaluateAST(f, data));
    else
        break;
}
SolverError_dump();
```

see file `SBML_odeSolver/tutorial/evaluateast.c`
The Sensitivity and Adjoint Solvers

OBJECTIVE FUNCTION

Construct Adjoint ODEs

EXPERIMENTAL DATA

SBML

Construct ODEs

ADJOINT

Adjoint solver

Parametric matrix

Sensitivity solver

ODEs

event detection

Forward solver

Jacobian matrix

time courses

Multiscale Modeling

(1) (2) (3) (4)

see Part II