ODEion - User Guide[*]

Peter Gennemark and Dag Wedelin

February 8, 2013

1 Introduction

ODEion is a software module for structural identification of ordinary differential equations (ODEs), where the model space is implicitly defined by arbitrary user-defined functions that can be non-linear in both variables and parameters, such as for example chemical rate reactions. The program searches for possible interactions between the variables and suggests a suitable form of the ODEs, including estimates of the parameters. The system has originally been developed for problems in systems biology but can be used for other applications. The name ODEion is an acronym for ODE IdentificatiON.

The program inputs a problem file in form of a standard text file. Input data manipulation is implemented in Java for flexibility, and time-critical code is generated in Fortran to allow for best performance and access to efficient numerical libraries. The Java Runtime Environment and a Fortran compiler are required. ODEion is distributed under the GNU general public license.

There are four basic functionalities:

- Verify syntax and consistency of a problem. Potential errors and reasonable solutions to these are reported.
- Visualize data from an experiment of a problem and adjusting parameters of the smoothing spline interpolation.
- Generate a Fortran program designed to solve a specific problem. The resulting program is self-contained, and can be compiled and run immediately, or saved for future execution.

[*]For formal reference please refer to Gennemark and Wedelin (submitted 2013). Software available at `http://odeidentification.org`.

- Generate a Systems Biology Markup Language (SBML) file of the solution model, allowing for visualization, simulation and further analysis on a range of softwares. Output is also given in a simple human readable format.

2 Program set-up

Requirements:

- Java Runtime Environment (version 1.6.0_17 used in program development).
- A Fortran compiler (gfortran, gcc version 4.4.4 used in development; http://gcc.gnu.org).

Besides, Perl (http://www.perl.org) may be useful for file manipulation.

Download the program to any directory on your computer. Go to the main directory ”ODEion_1_1”.

On a Windows system, it can be useful to work in the Cygwin environment where gfortran, the Java Runtime Environment and Perl are available. Cygwin is free of charge and can be downloaded at http://www.cygwin.com.

The Java program is pre-compiled. Potential recompiling of the program can be done by

```
javac Java/*.java
```

Documentation of the java classes contained in ODEion is available in the directory ODEion/Java/JavaDoc.

In the main directory of ODEion there is one library for problems, PROBLEMS, one library with sample scripts for generating new problems, GenerateProblem. The libraries starting by “F_” contain Fortran code, and the library OUTPUT is the default storage place for output files when running problems using one of the included scripts.

3 Running the program step by step

We will first describe how ODEion can be run step by step from the command line. In practice, it is useful to collect several calls in one single script, and this is described in the next section. For a quick start, readers that have Perl installed may choose to go directly to the next section.

Read a problem file and check its syntax

```
java Java/VerifyProblem aProblemFile
```

Several sample problems are available in the PROBLEMS directory. Example:

```
java Java/VerifyProblem PROBLEMS/simpleLin1
```

If the problem file is found feasible, only one single line is output, echoing the read file, e.g:

```
Java.problem.Problem.read - File: ....\ODEion\_1\_0\PROBLEMS\simpleLin1
```

To get additional debug output, add the flag -d as a second argument. Example:

```
java Java/VerifyProblem PROBLEMS/simpleLin1 -d
```

In this case, details of the reading and verification are reported on several output lines.

View time-series data of an experiment

```
java Java/PlotExperiment aProblemFile experiment\_number
```

For example, plot experiment 5 in the problem simpleLin2 by

```
java Java/PlotExperiment PROBLEMS/simpleLin2 5
```

A window should occur with the plot to the left and with check boxes to the right. Using the check boxes one can select which variables to view. Experimental data is interpolated by smoothing spline interpolation, and the effect of various smoothing parameters can be investigated by a slider in the window.

Create Fortran program for solving an identification problem

```
java Java/GenerateProgram aProblemFile
```

For example, generate a program for solving the problem simpleLin1 by

```
java Java/GenerateProgram PROBLEMS/simpleLin1
```

The generated code is saved in the directory F_generatedCode as the following Fortran files:

```
alg.f, alg_paraest.f, err.f, est.f, estvar.f, estvars.f, exp.f,
funjac1.f, funjac1general.f, funjac4.f, funjac4general.f,
main.f, model.f, reactions.f, readProblem.f, sim.f, spline.f
supportRoutine_dlsode.f, supportRoutine_dn2gb.f
supportRoutine_misc.f, supportRoutine_pppack.f
supportRoutine_zufall.f
```

The files with names starting by 'supportRoutine_' are constant for all problems, while all other files are problem specific. This can be confirmed by observing the modification times of the files by:

```
ls -l F_generatedCode
```

Compile and run Fortran program

Compile:

```
gfortran -o fprogram -w F_generatedCode/*.f
```

The flag -o is followed by the name of the executable program, and the flag -w indicates that no warning messages are printed.

Run:

```
./fprogram < PROBLEMS/simpleLin1
```

The solution file is saved as OUT_simpleLin1_1. This file is a standard human readable text file.

Program debug printouts are output on standard output, and can be captured, e.g., by:

```
./a.out < PROBLEMS/simpleLin1 > Run_simpleLin1_1
```

Create an SBML file from solution file

```
java Java/MakeSBML aProblemFile aSolutionFile theSBMLFile
```

Example:

```
java Java/MakeSBML PROBLEMS/simpleLin1 OUT_simpleLin1_1 OUT_simpleLin1.xml
```

The resulting SBML file can be simulated by standard SBML tools, see http://sbml.org. For example, an online service for verification and simulation: http://www.sys-bio.org/index.htm

4 Running the program from a script

We will now describe how ODEion can be run using a script that collects several calls.

A simple test script for one problem

A simple Perl test script for running problem simpleLin1 is given in the file script_test1.pl. It is executed by

```
perl script_test1.pl
```

The script can easily be modified to solve other problems.

The script executes four basic commands for (1) generating the Fortran program, (2) comiling the Fortran program, (3) running it, and finally (4) generating a SBML output file of the solution model. The individual calls are as follows:

```
java Java/GenerateProgram PROBLEMS/simpleLin1
gfortran -o F_program -w F_generatedCode/*.f
./F_program < PROBLEMS/simpleLin1 > Run_simpleLin1.txt
java Java/MakeSBML PROBLEMS/simpleLin1 OUT_simpleLin1_1  OUT_simpleLin1.xml
```

The perl script also reports some output in the following way:

```
$ perl script_test1.pl
Generate Fortran program.
 OK.
 Compile.
 OK.
 Run.
 OK
 FinalError =    8.0000...
 t_sec =    18.5...
 Create SBML file.
 OK
```

Here, FinalError refers to the value of the error function obtained for the solution model, and t_sec refers to the execution time in seconds (this problem should take less than 60 seconds on a standard computer).

A script for running several problems

A more advanced Perl test script for running several problems is given in the file script_test2.pl. Each problem may be run several times and with different seed to the random number generator. The script is executed by

```
perl script_test2.pl
```

By default, this script solves the problems SimpleLin1 and pe_3genes1. Each of the problems is executed with two different random seeds. Hence, in total there are four runs.

The output files for the two problems are stored in OUTPUT/RES_simpleLin1 and OUTPUT/RES_pe_3genes1, respectively.

It is easy to modify this script for user specific needs.

Partial derivatives

The program requires partial derivatives of the user-defined functions of $\mathcal{M}$, both with respect to the parameters θ and with respect to the variables X. By default, numerical derivatives are used and the user does *not* have to calculate and input any derivatives. However, the user can optionally define explicit derivatives for all functions of a problem. For this reason, a library of common functions (i.e., reaction kinetics) is maintained within the software in an ordinary text file: REACTIONS.txt. For example, the partial derivatives of reaction $\mathcal{M}_2$ from Eq. 2 in the paper is included in the library in the following form:

```
name = michaelisMenten
equation = k1*X1/(k2+X1)
dfdk1 = X1/(k2+X1)
dfdk2 = -k1*X1/(k2+X1)^2
dfdX1 = k1/(k2+X1)-k1*X1/(k2+X1)^2
```

Here, `dfdk1`, `dfdk2` and `dfdX1` correspond to the partial derivatives of the function, $\partial\mathcal{M}_3/\partial\theta_{21}$, $\partial\mathcal{M}_3/\partial\theta_{22}$, and $\partial\mathcal{M}_3/\partial X_k$, respectively. New functions can easily be added to the library using the same syntax. The use of explicit derivatives may improve accuracy of the solution and the computational speed to solve a problem.

5 Format used to specify the problems

This document details the format used to specify the benchmarks problems for identification of ordinary differential equations given at *http://www.odeidentification.org*.

We will first describe some general characteristics of the format, and then walk through the details of the identification file format, using the problem metabol2 as an example.

5.1 General structure of the format

With the exception of the begin..end construction at the beginning and the end of the file, the format consists of statements organized in paragraphs. A paragraph can consist of one or several statements. A typical paragraph has some similarity to a record or class in a programming language:

```
person_7
has name = mary
has telephoneNumber = 1234567
has child_ = person_5 person_14
is female
```

which can be understood as the single statements

```
name of person_7 = mary
telephoneNumber of person_7 = 1234567
child of person_7 = person_5 person_14
person_7 is female
```

In principle, every single statement is independent of every other statement, so the meaning of the statements is independent of their order. The indices used are all arbitrary and may be permuted. When a list is assigned (such as for child in the example above), it is assumed to end at the end of the line.

The particular format for identification problems follows this general structure, but has its own kinds of paragraphs, specialized to the purpose of this application. A simple reader for identification problems will then only be able to read these particular paragraphs for this application.

The format has been designed to be easy to parse in the following way. The syntax of a single paragraph is keyword based, in that you read the first identifier of the paragraph (before the '_', which is "person" in the example above), and then you know how to straightforwardly parse the entire paragraph. This can then be repeated over and over again until the end of the file.

All objects are referred to with a role and an index, eg. "variable_4". The indices are for each category consecutive beginning with 1. This means that you can easily read the file and directly place in suitable program data structures without the use of any hash tables. Dynamic arrays can however be useful, unless the file is read twice to check required array sizes. Not having such redundant size information in the file makes it easy to manually edit and modify the file without causing inconsistencies.

A simple example parser for identification problems is available on the web site. It is written in C and should be possible to compile and run on any computer with a C compiler. As it stands, it parses the identification problem file and prints it again. It can therefore be used directly as a simple verifier in that if the input and the output are identical, the file can be considered to be correct. When used in another program, the print statements can be replaced with code that loads the data into appropriate data structures.

5.2 Details about the format

We will now look at problem metabol2 in more detail. Our ambition is to make this documentation self-contained, but it can be useful to look at the full problem specification to get an impression of how all parts are combined. Finally, a general remark: '//' indicates that the rest of a line is a comment.

5.2.1 Header

```
format version = 1.0


begin problem metabol2

type = reactionKinetics
date =  5-Aug-08 09:27:04
url = http://www.cs.chalmers.se/~dag/identification/
```

The current and first format version is called 1.0. The problem is encapsulated by 'begin' and 'end'. 'problem' is a keyword followed by the name of the problem, which typically but not necessarily is the same as the name of the file.

There are two available types of problems: reactionKinetics and SSystem. Date and url specify when the problem was formulated and the address to the on-line information, respectively.

5.2.2 Reaction types

The reaction types specify the building blocks of the model space for a certain problem. Note that the model space is specified subsequently.

```
reaction_1
has name = uniMolecularMassAction
has localVariableName_1 = X1
has localParameterName_1 = k1
has equation = k1*X1

reaction_2
has name = biMolecularMassAction
has localVariableName_1 = X1
has localVariableName_2 = X2
has localParameterName_1 = k1
has equation = k1*X1*X2
```

```
reaction_3
has name = michaelisMenten
has localVariableName_1 = X1
has localParameterName_1 = k1
has localParameterName_2 = k2
has equation = k1*X1/(k2+X1)

reaction_4
has name = michaelisMentenNonCompInhib
has localVariableName_1 = X1
has localVariableName_2 = X2
has localParameterName_1 = k1
has localParameterName_2 = k2
has localParameterName_3 = k3
has equation = k1*X1/(k2 +X1)/(1+X2/k3)
```

A new reaction type begins with the keyword 'reaction_' followed by an integer. Each type has a unique name (used as reference in subsequent sections), a list of local variable names, a list of local parameter names and a rate equation. In the first example above, the rate equation is

$$k_1 X_1$$

where k_1 is a local parameter and X_1 is a local variable. In the second reaction type, 'biMolecularMassAction', the rate equations is

$$k_1 X_1 X_2$$

where k_1 is a local parameters and X_1 and X_2 are local variables. In the third reaction type, 'michaelisMenten', the rate equations is

$$\frac{k_1 X_1}{k_2 + X_1}$$

where k_1 and k_2 are local parameters and X_1 is a local variable. In the fourth reaction type, 'michaelisMentenNonCompInh', the rate equations is

$$\frac{k_1 X_1}{(k_2 + X_1)\left(1 + \frac{X_2}{k_3}\right)}$$

where k_1, k_2 and k_3 are local parameters and X_1 and X_2 are local variables.

5.2.3 Variables

Each variable has a name and is either an inputVariable or a dependent variable.

```
variable_1 has name = x1 is inputVariable
variable_2 has name = x2 is inputVariable
variable_3 has name = x3 is dependent
variable_4 has name = x4 is dependent
variable_5 has name = x5 is dependent
variable_6 has name = x6 is dependent
variable_7 has name = x7 is dependent
```

For each dependent variable there is a corresponding ODE. Hence, in the above example the system of ODEs to consider is:

$$\begin{aligned} x_3'(t) &= f_3(x_1, x_2, x_3, x_4, x_5, x_6, x_7, p, t) \\ x_4'(t) &= f_4(x_1, x_2, x_3, x_4, x_5, x_6, x_7, p, t) \\ x_5(t) &= f_5(x_1, x_2, x_3, x_4, x_5, x_6, x_7, p, t) \\ x_6'(t) &= f_6(x_1, x_2, x_3, x_4, x_5, x_6, x_7, p, t) \\ x_7'(t) &= f_7(x_1, x_2, x_3, x_4, x_5, x_6, x_7, p, t) \end{aligned}$$

where the f's are functions defined by the model space, p is a vector of parameters and t is time.

5.2.4 Ranges

A range is specified by a lower and an upper bound on the real line. Ranges are subsequently referred to when defining parameter bounds.

To specify a range we begin with the keyword 'range_' followed by an integer. Each range has a lower and upper bound.

```
range_1 has lowerBound =  0.000E+00 has upperBound =  0.500E+02
range_2 has lowerBound = -0.500E+02 has upperBound =  0.000E+00
range_3 has lowerBound =  0.100E+00 has upperBound =  0.500E+02
```

5.2.5 Variable sets

A variable set contains a set of variables. Variable sets are subsequently referred to when defining model space. A new variable set begins with the keyword 'memberOfSet_' followed by an integer, and finally a list of variables.

```
memberOfSet_1 variable_1 variable_2 variable_3 variable_4 variable_5
              variable_6 variable_7
memberOfSet_2 variable_4 variable_5 variable_6 variable_7
memberOfSet_3 variable_3 variable_5 variable_6 variable_7
memberOfSet_4 variable_3 variable_4 variable_6 variable_7
memberOfSet_5 variable_3 variable_4 variable_5 variable_7
memberOfSet_6 variable_3 variable_4 variable_5 variable_6
```

Note that the line break between 'variable_5' and 'variable_6' for 'memberOfSet_1' above is for display purpose and not part of the correct syntax.

5.2.6 Model space

The model space is defined separately for each dependent variable (ODE). Note that the model space may differ between the variables.

For each dependent variable there may be several possible reactions. A new possible reaction begins with the keyword possibleReaction_ followed by an integer, and then followed by 'of' and the variable to consider. Each possible reaction has a type (must be one of the pre-specified types above), a space of allowed variables for each local variable, and ranges for all parameters.

```
possibleReaction_1 of variable_3
has type = uniMolecularMassAction
has spaceOfVariable X1 = memberOfSet_2
has rangeOfParameter k1 = range_1

possibleReaction_2 of variable_3
has type = uniMolecularMassAction
has spaceOfVariable X1 = variable_3
has rangeOfParameter k1 = range_2

possibleReaction_3 of variable_3
has type = biMolecularMassAction
has spaceOfVariable X1 = memberOfSet_2
has spaceOfVariable X2 = memberOfSet_1
has rangeOfParameter k1 = range_1

possibleReaction_4 of variable_3
has type = biMolecularMassAction
has spaceOfVariable X1 = variable_3
has spaceOfVariable X2 = memberOfSet_1
has rangeOfParameter k1 = range_2

possibleReaction_5 of variable_3
has type = michaelisMenten
has spaceOfVariable X1 = memberOfSet_2
has rangeOfParameter k1 = range_1
has rangeOfParameter k2 = range_3

possibleReaction_6 of variable_3
has type = michaelisMenten
has spaceOfVariable X1 = variable_3
has rangeOfParameter k1 = range_2
has rangeOfParameter k2 = range_3

possibleReaction_7 of variable_3
has type = michaelisMentenNonCompInhib
has spaceOfVariable X1 = memberOfSet_2
has spaceOfVariable X2 = memberOfSet_1
has rangeOfParameter k1 = range_1
has rangeOfParameter k2 = range_3
has rangeOfParameter k3 = range_3

possibleReaction_8 of variable_3
has type = michaelisMentenNonCompInhib
has spaceOfVariable X1 = variable_3
has spaceOfVariable X2 = memberOfSet_1
has rangeOfParameter k1 = range_2
has rangeOfParameter k2 = range_3
has rangeOfParameter k3 = range_3
```

The above text specifies the possible reactions of variable 3. A summary is given in table 1.

Rate equation	**Variable space**	**Parameter range**
k_1X_1	$X_1 \in \{x_4, x_5, x_6, x_7\}$	$k_1 \in [0.0, 50.0]$
k_1X_1	$X_1 \in \{x_3\}$	$k_1 \in [-50.0, 0.0]$
$k_1X_1X_2$	$X_1 \in \{x_4, x_5, x_6, x_7\}$ $X_2 \in \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$	$k_1 \in [0.0, 50.0]$
$k_1X_1X_2$	$X_1 \in \{x_3\}$ $X_2 \in \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$	$k_1 \in [-50.0, 0.0]$
$\frac{k_1X_1}{k_2+X_1}$	$X_1 \in \{x_4, x_5, x_6, x_7\}$	$k_1 \in [0.0, 50.0]$ $k_2 \in [0.10, 50.0]$
$\frac{k_1X_1}{k_2+X_1}$	$X_1 \in \{x_3\}$	$k_1 \in [-50.0, 0.0]$ $k_2 \in [0.10, 50.0]$
$\frac{k_1X_1}{(k_2+X_1)\left(1+\frac{X_2}{k_3}\right)}$	$X_1 \in \{x_4, x_5, x_6, x_7\}$ $X_2 \in \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$	$k_1 \in [0.0, 50.0]$ $k_2 \in [0.10, 50.0]$ $k_3 \in [0.10, 50.0]$
$\frac{k_1X_1}{(k_2+X_1)\left(1+\frac{X_2}{k_3}\right)}$	$X_1 \in \{x_3\}$ $X_2 \in \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$	$k_1 \in [-50.0, 0.0]$ $k_2 \in [0.10, 50.0]$ $k_3 \in [0.10, 50.0]$

Table 1: *Summary of possible reactions of variable 3 in the problem metabol2*

Note that is assumed that $X_i \neq X_j$. For example, the following 'michaelisMenten-

NonCompInhib' reaction is not part of the above model space

$$\frac{k_1 X_3}{(k_2 + X_3)\left(1 + \frac{X_3}{k_3}\right)}$$

Also note that some combinations of the biMolecularMassAction reaction are equivalent, e.g. $k_1x_5x_6$ and $k_1x_6x_5$. Hence, in practice, one of them can be ignored. Table 2 gives the list of possible reactions explicitly written out (this time without ranges for the parameters).

k_1x_4	k_1x_5	k_1x_6	k_1x_7	k_1x_3
$k_1x_4x_1$	$k_1x_4x_2$	$k_1x_4x_5$	$k_1x_4x_6$	$k_1x_4x_7$
$k_1x_5x_1$	$k_1x_5x_2$	$k_1x_5x_6$	$k_1x_5x_7$	$k_1x_6x_1$
$k_1x_6x_2$	$k_1x_6x_7$	$k_1x_7x_1$	$k_1x_7x_2$	$k_1x_3x_1$
$k_1x_3x_2$	$k_1x_3x_4$	$k_1x_3x_5$	$k_1x_3x_6$	
$\frac{k_1x_4}{k_2+x_4}$	$\frac{k_1x_5}{k_2+x_5}$	$\frac{k_1x_6}{k_2+x_6}$	$\frac{k_1x_7}{k_2+x_7}$	$\frac{k_1x_3}{k_2+x_3}$
$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_1}{k_3}\right)}$	$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_2}{k_3}\right)}$	$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_3}{k_3}\right)}$	$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_5}{k_3}\right)}$	$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_6}{k_3}\right)}$
$\frac{k_1x_4}{(k_2+x_4)\left(1+\frac{x_7}{k_3}\right)}$	$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_1}{k_3}\right)}$	$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_2}{k_3}\right)}$	$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_3}{k_3}\right)}$	$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_4}{k_3}\right)}$
$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_6}{k_3}\right)}$	$\frac{k_1x_5}{(k_2+x_5)\left(1+\frac{x_7}{k_3}\right)}$	$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_1}{k_3}\right)}$	$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_2}{k_3}\right)}$	$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_3}{k_3}\right)}$
$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_4}{k_3}\right)}$	$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_5}{k_3}\right)}$	$\frac{k_1x_6}{(k_2+x_6)\left(1+\frac{x_7}{k_3}\right)}$	$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_1}{k_3}\right)}$	$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_2}{k_3}\right)}$
$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_3}{k_3}\right)}$	$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_4}{k_3}\right)}$	$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_5}{k_3}\right)}$	$\frac{k_1x_7}{(k_2+x_7)\left(1+\frac{x_6}{k_3}\right)}$	$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_1}{k_3}\right)}$
$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_2}{k_3}\right)}$	$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_4}{k_3}\right)}$	$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_5}{k_3}\right)}$	$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_6}{k_3}\right)}$	$\frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_7}{k_3}\right)}$

Table 2: *List of all possible reactions of variable 3 in the problem metabol2*

The ODE for variable 3 should be selected from any subset of the above 59 reaction terms. For the problem we consider as example, metabol2, the true ODE is

$$x_3'(t) = \frac{k_1x_3}{(k_2+x_3)\left(1+\frac{x_1}{k_3}\right)} + \frac{k_4x_3}{(k_5+x_3)\left(1+\frac{x_2}{k_6}\right)} + \frac{k_7x_4}{(k_8+x_4)} + \frac{k_9x_5}{(k_{10}+x_5)}$$

For S-systems, the model space is simply defined by bounds for the parameters.

```
alpha has defaultLowerBound =  0.

alpha has defaultUpperBound =  20.

beta has defaultLowerBound =  0.

beta has defaultUpperBound =  20.

g has defaultLowerBound =  -4.
g_1_1 has lowerBound =  0.
g_2_2 has lowerBound =  0.
g_3_3 has lowerBound =  0.

g has defaultUpperBound =   4.
g_1_1 has upperBound =  0.
g_2_2 has upperBound =  0.
g_3_3 has upperBound =  0.

h has defaultLowerBound =  -4.
h_1_1 has lowerBound =  1.E-15
h_2_2 has lowerBound =  1.E-15
h_3_3 has lowerBound =  1.E-15

h has defaultUpperBound =   4.
```

The keywords 'defaultLowerBound' and 'defaultUpperBound' can be used to assign one value to all elements in a vector or matrix. Naturally, subsequent assignments of specific elements overwrites the default value.

Besides, one of the benchmark problems include an additional constraint of type $\{g_{i,j} \in [-3, 3], g_{i,j} \neq 0\}$. This means that there is an interaction between variable i and j but the direction is unknown. We use the following syntax to constrain the value to non-zero:

```
g_2_1 is nonZero
h_5_5 is nonZero
```

5.2.7 Error function

The error function is defined by the keyword 'errorFunction'. The error function is generally defined as

$$-L(\hat{X}|\,\theta) + h\,(\lambda, K, N)\,. \tag{1}$$

The first term is the negative log-likelihood of the experimental data ($\hat{X}$ denotes experimental data, θ is a vector of parameters), and the second term penalizes structural complexity of the model (h is an arbitrary function, λ is a local parameter in the h function, K is the number of model parameters, and N is the number of data points). Common choices of h include AIC and BIC/MDL.

For identification of both structure and parameters we may, for instance, use the error function

$$-L(\hat{X}|\,\mathbf{k}) + \lambda K$$

coded as

```
errorFunction
has type = minusLogLikelihoodPlusLambdaK
has equation = -L+lambda*K
has lambda =   1.
```

Similarly, the error function

$$-L(\hat{X}|\,\mathbf{k}) + \lambda K log(N)$$

is coded as

```
errorFunction
has type = minusLogLikelihoodPlusLambdaKlogN
has equation = -L+lambda*K*log(N)
has lambda =   1.
```

For identification of parameters (fixed structure) one can use:

```
errorFunction
has type = minusLogLikelihood
has equation = -L
```

The parser in ODEion uses the information in the equation field to define the error function. The string in the type field is considered a name and is not parsed.

The log-likelihood is calculated by assuming independent and normally distributed measurement errors and disregarding constant terms. We express the log-likelihood for one time series as

$$L(\hat{X}_j|\,\mathbf{k}) = -\frac{1}{2}\sum_i \left(\frac{X_j(t_i) - \hat{X}_j(t_i)}{\sigma_j(t_i)}\right)^2$$

where i indexes the measurement points, and where X_j, $\hat{X}_j$ and σ_j denotes simulated data, experimental data and standard deviation for variable j, respectively. The total log-likelihood $L(\hat{X}|\,\mathbf{k})$ is defined by summing over all variables and all experiments.

For models based on chemical rate equations the number of parameters, K, is simply the total number of parameters on the right-hand side of the ODEs. For S-systems, it is natural to define K as the total number of non-zero elements in g and h plus the parameters in α and β.

5.2.8 Experiments

General information about the experiments are given in the following form.

```
experiment_1 has name = exp1
experiment_2 has name = exp2
experiment_3 has name = exp3
experiment_4 has name = exp4
```

Each experiment has a number and a name. For perfect data (no noise), 'has perfectData' can be added.

```
experiment_1 has name = exp1 has perfectData
experiment_2 has name = exp2 has perfectData
experiment_3 has name = exp3 has perfectData
experiment_4 has name = exp4 has perfectData
```

Then, details about each experiment are given.

```
sample_1 of experiment_1
has time =  0.0E+00
has variable_ =   0.200E+01  0.300E+02  0.1202588848674883E+01 ...
has sdev of variable_ =  0.0E+00 0.0E+00 0.53781408E-01 ...

sample_2 of experiment_1
has time =  0.75000000E+01
has variable_ =   0.200E+01  0.300E+02  0.9256634735714073E+01  ...
has sdev of variable_ =  0.0E+00 0.0E+00 0.92566347E+00 ...

 ...

21 of experiment_1
has time =  0.15000000E+03
has variable_ =   0.200E+01  0.300E+02  0.7581719037437853E+01  ...
has sdev of variable_ =  0.0E+00 0.0E+00 0.75817190E+00 ...
```

The keyword 'sample_' followed by an integer begins description of one sample. The rest of the line 'of experiment_1' states from which experiment the sample is taken. Each sample has a time-point, a vector of variables, and a vector of standard deviations. The vectors have the same length as the number of variables.

This compact form is useful when we have fully observed variables. In general, the sampled variables can be given individually as

```
sample_1 of experiment_1
has time =  0.0E+00
has variable_1 =   0.200E+01
has variable_2 =   0.300E+02
has variable_3 =   0.1202588848674883E+01
...
has sdev of variable_1 =  0.0E+00
has sdev of variable_2 =  0.0E+00
has sdev of variable_3 =  0.53781408E-01
...
```

5.2.9 Initial bounds

Initial bounds are given for each dependent variable in each experiments. The keywords 'variable_' and 'experiment_' followed by integers specify which variable and experiment to consider.

```
variable_3 of experiment_1
has lowerInitialBound =     0.1091E+01
has upperInitialBound =     0.1314E+01

variable_4 of experiment_1
has lowerInitialBound =     0.4505E+02
has upperInitialBound =     0.5388E+02

variable_5 of experiment_1
has lowerInitialBound =     0.4428E+02
has upperInitialBound =     0.5311E+02

variable_6 of experiment_1
has lowerInitialBound =     0.9109E+02
has upperInitialBound =     0.1087E+03

variable_7 of experiment_1
has lowerInitialBound =     0.1283E+01
has upperInitialBound =     0.1527E+01

variable_3 of experiment_2
has lowerInitialBound =     0.1139E+01
has upperInitialBound =     0.1363E+01
```

Note that information about initial bounds is unnecessary for perfect data.

5.2.10 Initial solution

The right-hand sides of all ODEs are by default assigned zero.

The initial solution specify a potential starting model other than zero. The syntax is similar to the one used to describe the model space. Instead of 'possibleReaction' we now write 'reaction', and instead of 'spaceOfVariable' we now write 'variable'. Here is an example for the case when the true structure is known for variable 3 of the problem 'metabol2'.

```
reaction_1 of variable_3
has type = michaelisMentenNonCompInhib
has variable X1 = variable_3
has variable X2 = variable_2
has rangeOfParameter k1 = range_2
has rangeOfParameter k2 = range_3
has rangeOfParameter k3 = range_3

reaction_2 of variable_3
has type = michaelisMentenNonCompInhib
has variable X1 = variable_3
has variable X2 = variable_1
has rangeOfParameter k1 = range_2
has rangeOfParameter k2 = range_3
has rangeOfParameter k3 = range_3

reaction_3 of variable_3
has type = michaelisMenten
has variable X1 = variable_4
has rangeOfParameter k1 = range_1
has rangeOfParameter k2 = range_3

reaction_4 of variable_3
has type = michaelisMenten
has variable X1 = variable_5
has rangeOfParameter k1 = range_1
has rangeOfParameter k2 = range_3
```

For S-systems, an initial solution is easily defined with similar syntax as used to define the model space. The keyword 'initialValue' is used to assign a specific element. The keyword 'defaultInitialValue' is used to assign all elements in a vector or matrix.

```
 alpha has defaultInitialValue = 1

 beta has defaultInitialValue = 1
 beta_2 has initialValue =  0.

 g has defaultInitialValue = 0
 g_2_3 has initialValue =  2.

 h has defaultInitialValue = 0
 h_1_1 has initialValue =  1.
 h_2_2 has initialValue =  1.
 h_3_3 has initialValue =  1.
```

For a problem with 3 dependent variables this gives:

$$\alpha = \left[\begin{array}{ccc} 1 & 1 & 1 \end{array} \right]$$

$$\beta = \left[\begin{array}{ccc} 1 & 0 & 1 \end{array} \right]$$

$$g = \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{array} \right]$$

$$h = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right].$$

5.2.11 End of problem

The problem ends in the following way:

```
end problem metabol2
```