SBSI Numerics User manual

Version 1.0.0

Authors: Richard Adams, Azusa Yamaguchi and Allan Clark

October 4, 2011

Contents

Acknowledgements

Funding

This work was supported by funding awarded to the Centre for Systems Biology at Edinburgh, a Centre for Integrative Systems Biology (CISB), by the BBSRC and EPSRC reference BB/D019621/1

Contributors

This software was developed initially by **Azusa Yamaguchi** and currently by **Allan Clark**. Many other people have contributed to this project, most of whom are listed below. If anyone who should be on is missing, we apologize, and please let us know so we can rectify the omission.

Anatoly Sorokin helped with the design and early development of the project.

Kevin Stratford, **Richard Adams**, **Carl Troein**, **Nikos Tsorman**, **Neil Hanlon** and **Shakir Ali** also made contributions to the code and documentation.

Galina Lebedeva provided example models and helped with testing the software.

Acknowledgments are also due to **Andrew Millar** and members of the Circadian Clock modelling group for providing direction, support and requirements to the project.

Liz Elliot and the CSBE admin team have provided a great deal of behind-the-scenes support to the project.

This project was led firstly by **Igor Goryanin** and currently by **Stephen Gilmore**.

For up-to-date information about help and contacts, please see our website:

`http://www.sbsi.ed.ac.uk`

Chapter 1

Introduction

1.1 Scope and aim of SBSI optimization framework

SBSI-Numerics is a software library and application designed to fit parameters for biological models to experimental data. As models become larger, optimization becomes increasingly unsuitable for running on desktop machines. SBSI-Numerics is parallelized and is designed to run on supercomputers or other parallel machines. So far SBSI-Numerics has been successfully installed on IBM BlueGene, HECTOR (the UK national supercomputer (Cray))and the Edinburgh Compute Data Facility (ECDF). However SBSI-Numerics can also be installed on Unix-based desktop machines - such as MacOSX, and Redhat scientific, Mandrake and Ubuntu flavours of Linux. SBSI-Numerics uses entirely open source third-party libraries, and is itself open-source. SBSI-Numerics accepts models in SBML level 2 and data in a standard data format (SBSI data format, described later in this document).

This document describes how to install the optimization framework and how to run the example files. The next chapters are devoted to the input file formats required. Finally, an in-depth description of the configuration is presented, along with some worked examples.

SBSI-Numerics is under continuous development and we will be interested to hear of your comments, bug reports and feature requests.

1.2 Optimization strategies supported by SBSI-Numerics

Optimization is the process of attempting to find the best possible solution amongst all those available. In the domain of this application, optimization attempts to find the best possible parameter values for a biological model to reproduce a given set of experimental data. The success of optimization depends on the choice of optimization algorithm, and also on the ability of the evaluation (a.k.a objective, cost) function to guide the optimization process towards a satisfactory solution.

The optimization framework seeks global optima for parameter values. During a *Setup* phase, all parameter space is searched using a quasi-random selection algorithm

which endeavours to search the parameter space uniformly to produce a starting parameter set. This starting set is then refined during the main optimization phase, which can use either simulated annealing or genetic algorithm techniques.

1.2.1 Genetic algorithms

Genetic algorithms (GAs) are search methods based on the principles of natural selection and genetics. GAs encode the decision variables of a search problem into strings of alphabets which represent candidate solutions to the problem. The strings are referred to as *chromosomes*, the alphabets are *genes* and the values of genes are termed *alleles*. In this application, a chromosome would represent a candidate set of parameter values to be evaluated. GAs rely on a *population* of chromosomes, which, over a number of generations undergo selection and mutation, where the evaluation function provides the selection force. The evolution proceeds through the following steps:

1. Initialization - an initial population of candidate solutions is generated (this is accomplished in SBSI-Numerics by the set up phase described in section 6.6.1).
2. Evaluation - the fitness values of candidate solutions are evaluated.
3. Selection - the best solutions are propagated to the next generation.
4. Recombination/mutation -these two processes allow the creation of novel parameter sets which may be better than the parental sets.
5. The novel sets replace the original parental population.
6. Steps 2-5 are repeated until some terminating condition is reached.

In SBSI-Numerics, this process is configurable. Configuration is described in detail in chapter 6 , but the major factors influencing outcome are:

1. The population size: Larger populations will cover more of parameter space, but will take longer to evaluate. Unnecessarily large populations may also duplicate search space and be wasteful.
2. The number of generations: More generations will give more opportunity for the search algorithm to reach the best, or target solution.
3. The number of parameter sets selected to seed the next generation will influence how the parameter sets evolve.
4. The mutation frequency will influence the search - high rates of mutation will produce different solutions, low mutation rates will produce new solutions more similar to previous solutions.

1.2.2 Simulated annealing

Simulated annealing techniques are based on an analogy with the physical annealing process of solids during cooling, where state transitions are effected by small perturbations, as the temperature is gradually lowered. In its usage in optimization, candidate solutions are equivalent to states of the physical system, and the cost of a solution is equivalent to the energy of a state.

Simulated annealing, as well as accepting improvements in cost, also accepts deterioration with a certain probability. This feature allows escape from local optima while still allowing iterative improvement.

In SBSI-Numerics's implementation, the major control variable for simulated annealing is the parameter `Mu` which controls how likely a worse solution is to be accepted. Increasing values of `Mu` decrease the probability of worse solutions being accepted to the next generation.

1.2.3 Objective (cost) functions

Chi-squared

The basic cost function supplied with the framework is the chi-squared function, which is evaluated for each data point in the experimental data:

$$(cal_y - dat_y)^2 * \frac{normFactor}{weight} \tag{1.1}$$

where cal_y is the simulation at that time point, and dat_y is the experimental measurement at that time point. For a data set, these individual costs are summed over all data points.

By default, normalization and weighting factors are 1. Future releases of SBSI-Numerics will allow encoding of weighting factors for experimental data points.

Chapter 2

Installation

This chapter explains how to set up SBSI-Numerics to run on your system. SBSI-Numerics can run on any Unix-based machine that has an MPI library installed. **This includes the Windows operating system if appropriate POSIX interfaces are installed, such as the CygWin package.**

2.1 Quick start automatic installation

The following sections detail the dependencies on which the SBSI numerics framework depends and how to obtain and install them on Mac and Linux based systems. However there is an automatic installation script which will on most systems take care of this for you. The script is named setup.py and is in the root folder of the SBSI distribution or can be obtained using the following link: `https://sourceforge.net/projects/sbsi/files/SBSINumerics/setup.py/download`

The script is written in the *python* programming language and hence requires that python is installed to run. If this is not already installed it can be obtained from `www.python.org`. This is a python version 2 script.

To invoke the script run the command:

```
$ python setup.py
```

This will print the help information which should guide you through installation of the SBSI numerics framework and dependencies.

An important point however is that the installation directory for SBSI and dependencies can be set using the option `install=/some/directory` as in the command:

```
$ python setup.py all install=/usr/
```

The default is to install to a directory named "install" under the current working directory. If this, or whatever installation directory is not a default location then in order to use the SBSI framework the installation directory must be added to various

system paths and compiler search paths. This can be using the script found at: `https://sourceforge.net/projects/sbsi/files/SBSINumerics/set-vars.sh/download`

This can be invoked with the command:

```
$ source set-vars.sh
```

or

```
$ chmod u+x set-vars.sh
$ ./set-vars.sh
```

In either case the installation directory is assumed to be a directory named "install" under the current directory but can be specified as the only command line argument as in:

```
$ source set-vars.sh /my/install/directory
```

2.2 General requirements

Version 4 or later of the GNU C++ compiler and GNU make are recommended if it is available. The "make install" command installs libraries in standard places, and typically requires root privileges, unless you specify the place with the –prefix flag to configure. E.g., to install into a non-standard location:

```
./configure  --prefix=/my/preferred/install/folder
```

SBSI-Numerics depends on the following libraries. The versions needed and where to obtain them are explained in more detail in future sections.

1. **libsbml** At least level 2 version 3 support, 3.4.0 or greater.
   This is optional. If you do not have to generate C++ code for SBSI-Numeric on your system, SBSI-Numeric will be built without libsbml with the flag "–with-sbml=no" to configure.

2. **libxml2**
   This is an XML parsing library required by libsbml. Most Linux distributions include this package.

3. **MPI** Open MPI, MPICH or LAM/MPI.
   Since MPI-2 standard has a bug for definition of SEEK_SET, SEEK_END, SEEK_CUR, if you use MPICH, you need to add a flag, "CXXFLAGS=-DMPICH_IGNORE_CXX_SEEK" to configure.

4. **Sundials Library v2.3.0** differential equation solver.
   The newest version of Sundials is v2.4.0, however SBSI-Numeric uses a modified v2.3.0 with prefetch optimisations added by SBSI-Numeric author. v.2.4.0 is not compatible.

5. **PGAPack** A parallelized optimisation library for Genetic Algorithm.
   It was developed by David Levine of the Mathematics and Computer Science Division at Argonne National Laboratory. We have fixed bugs and changed its header files to run with C++ compiler, so use PGAPack included in SBSI-Numeric distribution.

6. **FFTW** Fast Fourier Transform library used for FFT cost functions.
   There are two FFTW version SBSI-Numeric supports, although the API of FFTW 3.x and that of FFTW 2.x are not compatible. FFTW 2.x supports MPI for distributed-memory, and FFTW 3.x does have support for Cell processors but not MPI.

7. **CppUnit** This is optional. If you want to run unit tests, Installation of libcppunit is required.

2.3 Installation on MacOSX

To install SBSI-Numeric, your system has to have the Apple developer tools installed.
MacOS system needs

```
-Wl,-search_paths_first
```

to use newly created libraries instead of system libraries. If you wish to override system libraries with new versions, pass this flag to configure.

Installing libFFTW

For FFTW v3.x,

```
./configure --disable-fortran CC=gcc CXX=g++ CFLAGS=-O3 CXXFLAGS=-Wall
make
make install
```

For FFTW v2.x,

```
./configure --disable-fortran --enable-type-prefix CC=gcc CXX=g++ CFLAGS=-O3 CXXFLAGS=
make
make install
```

Installing libxml2

The standard simple commands are enough.

```
./configure
make
make install
```

The minimal requirement for SBSI-Numeric for libxml2 are available to be built with the command below.

```
./configure CC=gcc CXX=g++ CFLAGS=-O3 --without-zlib --without-python --without-readli
make
make install
```

Installing libSBML

As the same as libxml2, the standard commands will build the library.

```
./configure
make
make install
```

This configure command will find the libxml2 in standard places, /usr/local/lib or /usr/lib . If you want to specify the place of the libxml2 to build the libSBML, the flag –with-libxml2 to configure will find that one.

The minimal requirement for SBSI-Numeric is available to build with,

```
./configure  CXX=g++ CFLAGS=-O3 CXXFLAGS=-Wall --without-zlib --without-bzip2
make
make install
```

Installing PGAPack

```
./configure CC=mpicc CFLAGS="-O3 -lm"
make
make install
```

Installing Sundials

As the same as libxml2, the standard commands will build the library.

```
./configure
make
make install
```

If your system does not support Fortran, use –disable-f77 flag to configure.

Installing SBSI-Numeric

Now we have installed all the dependent libraries and can install SBSI-Numeric!

```
./configure CXX=mpic++ CFLAGS=-O3 CXXFLAGS=-Wall
 make
 make install
```

The configure command will find the libxml2, libsbml, libFFTW v3.x, libsundials, libpgapack under the standard directory, /usr or /usr/local. If you specify the libraries place, use –with-libxml2, –with-sundials, –with-pgapack –with-fftw3 flags to configure. SBSI-Numeric try to find FFTW v3.x as the standard FFTW library. If you want to use FFTW v2.x, the flags –with-fftw3=no –with-fftw2 will make available to link fftw v2.x instead of v3.x. If the system does not have libSBML, SBSI-Numeric will skip to codes which has dependency of SBML.

E.g., here is an example configure invocation where all the installed libraries have their locations specified.

```
./configure CXX=mpic++ CXXFLAGS="-O3 -Wall " --with-libxml2=/Desktop/SBSINumReleases/inst
--with-sbml=/Desktop/SBSINumReleases/install  \
--with-sundials=/usr/local \
--with-pgapack=/Desktop/SBSINumReleases/install \
--with-fftw3=/Desktop/SBSINumReleases/install \
--prefix=/Desktop/SBSINumReleases/install \
```

We recommend to "make check" after built SBSI-Numeric before make install.

Checking your installation works

```
make check
```

2.4 Installation on Linux

Installing libFFTW

As well as installing from source, the packages from Debian and Redhat are available from their home page,

`http://www.fftw.org/install/linux.html.`

Installing libxml2

The standard simple commands are enough.

```
./configure
 make
 make install
```

The minimal requirement for SBSI-Numeric for libxml2 are available to be built with the command below.

```
./configure CC=gcc CXX=g++ CFLAGS=-O3 --without-zlib --without-python --without-readli
 make
 make install
```

Installing libSBML

As the same as libxml2, the standard commands will build the library.

```
./configure
 make
 make install
```

This configure command will find the libxml2 in standard places, /usr/local/lib or /usr/lib . If you want to specify the place of the libxml2 to build the libSBML, the flag –with-libxml2 to configure will find that one.

The minimal requirement for SBSI-Numeric is available to build with,

```
./configure  CXX=g++ CFLAGS=-O3 CXXFLAGS=-Wall --without-zlib --without-bzip2
 make
 make install
```

Installing PGAPack

```
 ./configure CC=mpicc CFLAGS="-O3 -lm"
make
make install
```

Installing Sundials

As the same as libxml2, the standard commands will build the library.

```
./configure
 make
 make install
```

If your system does not support Fortran, use –disable-f77 flag to configure.

Installing SBSI-Numeric

```
./configure CXX=mpic++ CFLAGS=-O3 CXXFLAGS=-Wall
 make
 make install
```

The configure command will find the libxml2, libsbml, libFFTW v3.x, libsundials, libpgapack under the standard directory, /usr or /usr/local.
If you specify the libraries place, use –with-libxml2, –with-sundials, –with-pgapack –with-fftw3 flags to configure.
SBSI-Numeric tries to find FFTW v3.x as the standard FFTW library.
If you want to use FFTW v2.x, the flags –with-fftw3=no –with-fftw2 will make available to link fftw v2.x instead of v3.x. If the system does not have libSBML, SBSI-Numeric will skip to codes which has dependency of SBML. We will recommend to "make check" after building SBSI-Numeric before make install.

Now we have installed all the dependent libraries and can install SBSI-Numeric!

Checking your installation works

```
make check
```

Chapter 3

Input requirements

This chapter describes the range of SBML supported by SBSINumerics, and how constraints on parameters are supplied to the framework.

3.1 Introduction

SBSINumerics provides a utility, SBML2C, that will convert SBML to C++. SBML2C will work with an valid SBML model that is level 2. Currently SBSINumerics uses level 2 version 4 as its preferred version - previous versions of level 2 will be updated to version 4. The one absolute requirement for the model is that the `Model` element has a valid 'id' attribute. The model ID is used internally by SBSINumerics throughout the optimization process.

If the SBML file has errors, or cannot be updated to level 2 version 4, then conversion will fail.

3.2 Supported SBML subsets

SBML2C is broadly supportive of libSBML level 2, and supports rate rules, kinetic laws, reactions, species, function definitions, events and compartments. For models that contain identifiers that are C++ reserved words, the tool will replace variable occurrences with the variable name appended with "_MODEL". Typically the main culprits are compartments called 'default'.

3.3 Unsupported SBML subsets

The following elements are not specifically handled yet :

1. Delays in event assignments
2. Constraints are not handled, as they are supplied in a separate file called [modelID]_InitParam.dat

3. Piecewise elements in Math functions, unless a piecewise element is the top level math element in an assignment rule.(I.e., a conditional assignment).

4. The 'power' function can be ambiguous if both arguments are integers and may cause the model compilation to fail.

In terms of the math supported, SBML2C will generate C++ code for any math that is correctly specified for its enclosing SBML element. However, not all math functions supported by SBML's specification have standard C++ functions. The following is a list of math functions that will generate C code that will not currently compile with the standard C++ math library:

1. Inverse trig functions such as *sec*, *cosec*, *cot*, *arcsec*, *arccosec*, *arccot* and inverses of hyperbolic trig functions such as *arccosh*, *arcsinh*, *arctanh*.

2. Logarithms to bases other than e or 10.

3. Roots other than square roots.

3.4 Parameter constraint files

Parameter constraint files should be named following the convention 'ModelID_InitParam.dat', where 'ModelID' is the SBML model ID.

So, for example, for a model with SBML id 'abc_1', its parameter constraint file would be named 'abc_1_InitParam.dat'.

The format is a simple tab-delimited format. If you are optimizing using the genetic algorithm, the format is:

[Parameter ID] [Min] [Max]

As an example:

```
K1_reaction4 0.0 0.5
K2_reaction5 0.0 0.5
K3_reaction6 0.0 0.5
K4_reaction7 0.0 0.5
```

For simulated annealing, an additional column is needed for the initial step size. E.g.,

```
K1_reaction4 0.0 0.5 5.0E-4
K2_reaction5 0.0 0.5 5.0E-4
K3_reaction6 0.0 0.5 5.0E-4
K4_reaction7 0.0 0.5 5.0E-4
```

A candidate step size is 1/100th of the range between the min and max parameter values.

In order to be valid, the following inequalities should be satisfied for each row (parameter constraint):

$min < max$

and (for simulated annealing):

$min + stepSize < max$

Chapter 4

SBSI Data Format Specification

4.1 General Comments

Many parameter optimisations will require experimental data. This chapter describes the data format currently used within SBSI, called SBSI data format.

SBSI data format is a file format used for experimental and simulation data. SBSI data format uses a bipartite file structure where the data is stored in essentially a tab-delimited format, and annotation is stored in a separate *header* file. Data files refer to header files; a header file can be referenced by many data files.

In all files comments are indicated by line starting with a hash (#) character. Empty lines (pure white-space) are allowed to aid clarity.

The header file defines the columns in the data file so every data file must refer to an existing header file to be valid. Several data files can refer to the same header file.

A data file may refer to a subset of the columns defined in the header.

The header file can provide simple annotation of an experiment that the data files refer to. Trailing tabs at ends of lines should also be avoided, in both files.

4.2 Header File

The header file is organised into seven sections as follows:

- *Annotation*
- *InitialValues*
- *Parameters*
- *Columns*
- *X2datasetconfig* - optional, may be absent
- *FFTdatasetconfig* - optional, may be absent

- *StatesTable* - optional, may be absent. Mandatory if FFTdatasetconfig section has data in it.

where the * delimiters denote a section. The order of the sections is unimportant. The first four of these elements are part of the 'core' data format, the latter three are used for the convenience of tools wishing to persist configuration information.

All the elements in a header file are optional. A minimal header file is:

```
*Annotation*
*InitialValues*
*Parameters*
*Column*
```

4.2.1 Annotation

This enables the description of any experiment wide annotation. It is a single line of the form:

< annotation field name > < description >

(In this and subsequent element descriptions, the < and > are just used to demarcate field names, but are not used in the files themselves). The section is started by the following heading:

Annotation

4.2.2 Initial Values

This section can be empty and provides a set of initial values that may be used in a model associated with this experimental data. It is a single line of the form:

<species name> <units> <value>

Note that each item is delimited by white space.

The section is started by the following heading:

Initial Values

4.2.3 Parameters

This section can be empty and provides a set of initial parameter values to be used in model associated with the experimental data associated with this header. Each entry is defined on a single line.

<parameter name> <units> <value>

Each item is delimited by white space.

The section is started by the following header:

Parameters

4.2.4 Column Definition

This defines the columns present in the data file.

<column name> <units> <description>

Column names must be unique. Column names are compared in a case-insensitive manner so, for example, cyclin_A and CYCLIN_A are considered duplicates.

The section is started by the following heading:

Columns

4.2.5 X2DataSetConfig

This defines values to be used in X2Costfunctions. Each row of data entered refers to a specific data file. Many rows of data may be present as many sbsidata files may share the same header. The values are as follows:

- Filename, start time, scale type, value type, use in setup, interval.
- Values can be comma OR tab separated.
- The File name must not be empty.
- Start time and interval should be doubles, neither may be negative and interval may not be zero.
- Scale type must be one of : None, Average, AbsAverage, MinMax, Fix
- Value type must be one of: Normed, Direct
- Use in setup must be true/false.

4.2.6 FFTDataSetConfig

This defines values to be used in FFTCostfunctions. Only 1 row of data can ever be present as a single FFT Cost Function is defined per header.

The values are as follows:

`use in setup`, `interval`.

Values can be comma OR tab separated.

`Use in setup` must be true/false.

`Interval` should be a double > zero.

4.2.7 StatesTable

Every state/species in the SBML model must have an entry here if an FFTCostFunction is being configured. There may be multiple rows of data.

The values are as follows:

`Use`, `name`, `start time`, `period`.

Values can be comma OR tab separated.

`Use` must be true/false.
`Name` must not be empty.
`Start time` must be a positive double.
`Period` must be a positive double > 0.

4.3 Data File

The data file must refer to its header and consists of the data to be used. It has the following structure:

Header Reference
Column Declaration
Data Values

4.3.1 Header Reference

This is a reference to the file containing the header definition. This should be given as a UNIX style file path (even on Windows) and can be either a relative or absolute path (relative is generally more portable and therefore preferred).

This consists of a line with the following format:

@!<header path>

Note that the @! must be the first 2 characters of the file and the path should follow without any space between it and its first character. Note that the path can contain spaces as the file path is read from character 3 until the last non-white space character on the line.

4.3.2 Column Declaration

This contains one or more column names defined in the header. They must be unique (duplicate column names cannot be used). Column names are compared in a case-insensitive manner e.g., cyclin_A and CYCLIN_A are considered duplicates. The column declaration can only span one line.

The format is:

<column name>.

4.3.3 Data Values

These are one or more fields that correspond to the column declaration. The data values effectively define a table, and so there must be one value in every row for column that appears in the Column Declaration. Missing values should be indicated by the value null.

<value>..

4.3.4 Lexical Notes

Fields are separated by any white space characters (note that the Header reference is a special case).

Valid field definitions are defined below:

- Column Name [A-Za-z][A-Za-z0-9_-]* foo_bar is valid, but foo bar is not, nor is 1foo_bar. No spaces are allowed within the name.
- Description [A-Za-z]*.*$ Can have spaces and is terminated by the end of the line.
- Header Path. See notes above
- Units [A-Za-z][A-Za-z0-9-()_$\hat{]}$* This should conform to a controlled vocabulary. Examples are ms^{-2}.
- Value [-+]?[0-9]*$\dot{?}$[0-9]+([eE][-+]?[0-9]+)? Matches integer, floating point and scientific notation in form 1.5e12 or 1.5E12
- Species Name as Column Name
- Parameter Name as Column Name

4.3.5 Example

This is an example of a header file

```
Header File
*Annotation*
Example header file for simple biochemical system (ABC)
synthetic data generated for Initial values A=950; B=20; C=30,
using abc_1.slv (abc_1.xml), all parameters equal 1, simulation time 10 s

# These initial values override those described in the SBML model.
*Initial values*
A nm 950
B nm 20
C nm 30

# These parameters override those described in the SBML model.
*Parameters*
k1 n/a 1.5

#Annotation and units of columns
*Columns*
T s time
A nm substrate concentration
```

```
B nm intermediate product
C nm product
```

and below is a sample of a data file that refers to this header file. Note the agreement between column headings in the data file, and column names in the 'Columns' section of the header file.

```
@!ABC_9502030.sbsiheader
T A B C
0 950 20 30
0.000387784 949.6396376 20.35247 30.00789235
0.000775567 949.2795514 20.70452733 30.01592123
0.001551135 948.5599306 21.40781776 30.0322516
```

Chapter 5

Running the framework

5.1 General requirements

To use the framework from the command line, you need to have a directory that has the following resources:

- An SBML model.xml file,
- A model.dat data file in SBSI data format
- A model.hdr file in SBSI data format
- An xml configuration file
- loop.sh - this is the main executable shell script
- main_Model.C- C file containing the main executable
- Makefile, copied from the shared/sbsi folder of your installation.

In the installation, an examples/ folder contains several example projects which illustrate how the process works. The configuration files can be used as templates to begin your own configuration files and are explained further in the section on configuration.

In summary, the steps are :

1. Run the SBML2C utility tool, with your model file as an argument. This will generate your model.C file from SBML.
2. Compile your model.C file.
3. Execute SBSI-Numeric.

These are explained further below.

5.2 Optimizing a model

In this section we go through the steps needed to run a real model.

Assume we have a folder called 'Clock' with a model called Clock.xml, and data called clock.dat. This folder can be anywhere on your file system. Assume you have previously installed SBSINumerics . The folder (which we'll call the *job folder* from now on) should contain at least the following files:

1. Clock.xml - your model file in SBML format, level 2. It is essential that the model has an 'id' attribute. I.e., the top level model element looks something like: <model id="mymodel">

2. clock.dat - your data file, in SBSI data format.

3. clock.hdr, a header file annotating the data file, in SBSI data format.

4. A file called modelID_InitParam.dat which contains the constraint information for the parameters (i.e., the max and minimum values). In your file name, you should replace the characters 'modelID' with the SBML model id of your own model. This file should be in the following, tab delimited format :

   For genetic algorithm:

   ParamID minValue maxValue

   For simulated annealing:

   ParamID minValue maxValue initialStep

   You only need to list parameters that you want optimised. For PGA optimisation, at least 3 parameters must be chosen.

   For an example of this format see 3.4.

Now, copy the the following files can from the /shared/sbsi folder where you installed it.

- loop.sh - the main executable script.
- Makefile - for compiling your model file. This should not need to be altered.
- main_Model.C

- Now, at a terminal window:

```
cd ./Clock
/path/toSBML2C Clock.xml
cp UserModel\_InitParam.dat ./UserModel
make MODEL=Clock
```

(Replace the word 'Clock' with your own model ID (the value of the 'id' attribute of the 'Model' element in your sbml file)). Check - at this stage, if you look in the job folder, there should now be a subfolder named "UserModel". Inside this subfolder, you should find .C, .h and .o files for your model, as well as the initParam.dat file. It should have the following structure.

Before running, the folder system should look like this, with the following files:

```
/jobFolder
    /UserModel
        /results
        /exp_data
        /ckpoint
        MyModel_InitParam.dat
        UserModel.C
        UserModel.h
    Makefile
    MyModel.exe
    loop.sh
    config.xml
    main_MODEL.C
```

Now, run

```
./loop.sh Clock config.xml
```

loop.sh takes two arguments - the modelID and the configuration file name. Configuration is explained in chapter 6.

This should start the optimisation program running.

5.3 Known issues

There are several steps where things can potentially go wrong. Here are some known issues which cause SBSI-Numeric to fail.

1. Correct data format - the data files must be plain text, tab-delimited, with no control or special characters. Sometimes export from a spreadsheet can include these characters. To check this, open your data file in an editor such as vim. If the data does not appear correctly and 'M̂' characters are seen, you can remove them by performing a substitution: In vim, type : %s/CtrlV CtrlM//g (on Mac) to format the data properly. - also, data and header files should not have trailing empty tabs at the end of each line. For more information on SBSI data format see chapter 4

2. Correct format of init_Param.dat This file needs to be supplied by the user, and is generated automatically by SBSIVisual. For more details, see section 3.4

Only parameters to be optimised should be in this file. Parameters defined locally inside an SBML kineticLaw or reaction should be identified by paramID_reactionID to ensure parameters are uniquely identified.

Invoking the framework will fail unless at least 3 parameters are supplied in the parameter file.

5.4 Results

The results of an optimization run are put in the UserModel/results/ subfolder of the job folder. Some of the result files are listed here:

5.4.1 PGALog

This contains the full output of the optimization framework. Every parameter value is output for every parameter set at intervals of generations specified by the `Verbose` configuration element. For example, if `Verbose` = 4, parameter sets are output at generations 4,8,12, etc.,

5.4.2 PGA.StopInfo

This file describes the reason the optimization framework stopped (see chapter 6 for more details on possible reasons for stopping).

5.4.3 BestTSeries...

Files starting with this prefix contain a time series of a simulation using the best parameters found during optimization. A time series is produced for each data set being optimized, to take account of initial conditions.

5.4.4 BestCost..

These files contain the best parameter values found, and the value of the cost function.

5.4.5 Error and log files

Files reporting progress can be found in the top-level job folder. For example, output.Log contains all information emitted to stdout during the optimization run. If a file called OFW.Error is present, this indicates that something has gone wrong during optimization, and contains a brief explanation of the reason for failure.

Chapter 6

Configuration of SBSI -Numeric optimization framework

6.1 Introduction

This chapter describes the user configurable parameters for running the SBSI-Numeric optimization framework. These can be broadly classified into several main areas: configuration of the ODE solver, configuration of the optimization algorithm and configuration of the cost function and stopping conditions. A complete list of configurable parameters, default values and detailed explanation is given in 6.6.2 below.

This document describes configuration from the command line - client tools such as SBSIVisual may provide validation and defaults to ease configuration.

In this chapter, elements that appear in the XML configuration file appear in `typewriter font`.

6.2 Location of files

Each optimization job requires several resources: model files, template C files etc., This is covered in the chapter on running SBSI-Numeric 5.

Before starting, the jobFolder should contain all the resources you need, including those from the shared/sbsi subfolder of your installation folder. By running SBML2C, followed by compilation of the model, and running :

```
loop.sh modelID config.xml
```

the folder structures will be auto-generated and only initParam.dat needs to be copied into the model folder.

In the configuration below, filenames can be given just as names, with no folder path prefix, if they are initially in the jobFolder.

6.3 Overview of the optimization process

Configuration is enabled both for running optimizations locally and on HPC machines. In the latter case, or for long running jobs, it is often useful to have a series of checkpoints which give interim results. This gives the user the opportunity to restart an optimization from an intermediate point, with an altered configuration, in order to achieve better results without having to begin the optimization from the beginning.

Optimization proceeds using an objective, or cost function to evaluate the goodness of fit of a particular parameter set. In this release, Chi-squared cost function (for evaluating the fit between simulation and experimental data) and FFT cost function (for evaluating the periodicity of oscillatory models) are available to use.

6.4 Stopping criteria

The following criteria are used to determine how to stop an optimization job, started by a user. In general the user will supply a target cost function value, which based on their domain knowledge will provide a suitable level of parameter optimization: The job will terminate if any one of the conditions below is met.

- An absolute number of generations is reached (`MaxNumGen`). This is a back-stop for pathological cases in which subsequent stop criteria are insufficient.
- The target cost function value, `TargetCostFunctionValue`, is reached.
- A number of generations, `MaxNoChange`, is reached, with no improvement in the costfunction.
- The percentage similarity of the population exceeds the user defined value `MaxSimilarity`.

The user can further refine the stopping behaviour by use of the parameters `NumGen` and `RestartFreq`. `RestartFreq` defines a number of generations after which the population will be replaced if there is no improvement in the cost function. This will need to be a value < `MaxNoChange`. If, despite replenishing the population, there is no improvement within `MaxNoChange` generations then SBSI-Numeric will still terminate.

In order to minimize waiting times on Bluegene or other HPC systems, a single large job can be split into several smaller jobs which are linked together. This can be configured by setting `NumGen` to a fraction of `MaxNumGen`.

E.g., NumGen = 1000, MaxNumGen=8000 will enable 8 separate 'loops' of optimization to be performed.

After each loop, results and parameter values are output to the results folder, and the count for `MaxNoChange` and `RestartFreq` values are reset to 0. This means the following criteria need to be met for a job involving looping, where restarts are required:

1. `NumGen` < `MaxNumGen` (Possible settings could be: numGen=1000, maxGen=8000). `NumGen` is the number of generations in a loop.

2. For optimization to stop due to no improvement in cost function, `MaxNoChange` < `NumGen`. If `MaxNoChange` > `NumGen`, optimization will not stop even if the cost function fails to improve.

3. If restarts are desired (these refresh the population of parameter sets):

   - `RestartFreq` < `NumGen` (else a loop will be finished without triggering a restart)
   - `RestartFreq` < `MaxNoChange` (else optimization will terminate due to `MaxNoChange` being reached, before `RestartFreq` is reached)
   - `MaxNoChange` < `NumGen` (else optimization will keep restarting even if cost function stops improving).

An example combination of settings might be: `NumGen` = 500 `RestartFreq` =250 `MaxNoChange`=400

In this scenario, using the above settings:

If, from generation 0, cost function value fails to improve over 250 generations, a restart will be triggered with a fresh population.

If, despite the restart, cost function value does not improve in generations 250-400, optimization will terminate at generation 400. Or, if cost function value does begin improving again from 250, optimization will continue until the generation 500. At this stage, if `MaxNumGen` has been exceeded, optimization would terminate. Otherwise, another loop will start. If at any point in this scenario, the target cost function is reached during this time, then optimization will terminate.

After each `NumGen` generations, while SBSI-Numeric continues to run, it will output a time series of the model with the current best parameter values. So for example, if `NumGen`=100, and `MaxGen`=1000, then 10 sets of interim results would be generated, after 100,200,300 etc., generations (provided, of course that none of the other stop criteria are reached).

For long jobs, especially running remotely, this can be a useful feature.

For local, small or test jobs, the above configuration criteria can be greatly simplified by:

1. Set `MaxNumGen`=`NumGen` - this prevents looping

2. Set restartFreq = `NumGen` +1 - this prevents restarts

3. Set `MaxNoChange` < `NumGen`. This will enable stopping due to failing to improve cost function criteria.

In other words, SBSI-Numeric will continue so long as the cost function is decreasing - achieved by using a checkpoint mechanism. The basic `NumGen` parameter will now be the number of generations between checkpoints.

6.5 Reporting results

At termination, SBSI-Numeric will report the reason for stopping, and state of all relevant variables determining the stop criteria above, to help the user refine SBSI-Numeric parameters for the next job, in the event the target cost function was not reached. SBSI-Numeric is deemed to be terminated when either:

1. For a successful job, a line beginning with FINISH appears at the end of the file OFW.status
2. For a failed job, a file, OFW.error is generated containing a record of the errors.

All results are saved in the 'results' folder of the modelID/ subfolder of the project folder.

6.6 Configurable parameters

Configuration of the optimization framework is achieved through an XML configuration file. Here we describe the elements of the XML file.

6.6.1 Configuration of the `Setup` phase

. The setup phase is an initial phase of the optimization process where the initial parameter values are determined for the first round of optimization.

- There are two main options, specified by the `SetupType` element, which is mandatory.
  - Read in an existing checkpoint file from a previous run of the optimization framework.(ReadinFile). This is used in conjunction with the `CkpointNum` element which specifies the number of a checkpoint file in the UserModel/ckpoint subfolder. For example,

    ```
    <SetupType> ReadinFile </SetupType>
    <CkpointNum>23</CkpointNum>
    ```

  - Use an algorithmic approach to identify a good starting set of parameter values(`SobolSelect` or `SobolUnselect`). The former uses the Sobol quasi-random number generator to sample parameter space evenly, and to be accepted, each parameter set must evaluate to less than `MaxCost`. `SobolUnselect` merely generates the parameter sets but does not select based on `MaxCost`.
- `MaxCost` is an element required if `SobolSelect` is chosen. During the setup phase, a parameter set with cost larger than `MaxCost` cannot be in the first generation of GA. This means that the lower the value of `MaxCost`, the longer the set-up phase. Default - 1000? (allows quick testing of a model).

- `ParameterInitialFile` is a mandatory element whose value is the file name of the parameter constraint file (e.g., 'myModel_InitParam.dat').

- `CkpointNum` is a mandatory element if the SetupType is 'ReadinFile', otherwise it is not needed. The value of this element should be an integer present in the name of a checkpoint file. Checkpoint files are named in a syntax like 'BestPop.297.ga'. If we wanted to use this checkpoint file the element would be

  ```
  $<$CkpointNum$>$297$<$/CkpointNum$>$
  ```

- `MaxRandomNumberGenerator` The maximum number of Sobol points generated, if `SobolSelect` is chosen. This should be an integer value, usually set to 5000-10000. This is the maximum number of times that a random parameter set will be generated and evaluated, and tested to be less than `MaxCost` This value MUST be bigger than the population size, as each parameter set in the starting population is generated independently. So, for example if popsize is 1024, we would need `MaxRandomNumberGenerator` to be at least 1024. This is an absolute minimum, as optimization would only proceed if every randomly generated parameter set results in a cost less than `MaxCost`. In practice this would be set much higher than 1024.

- `ModelCostFunction` This element is described fully in the CostFunction configuration section, but at least one element of this type is needed in the setup phase in order to evaluate the initial parameter sets.

6.6.2 Configuring the optimization process

Configuration of the optimizer is achieved by sub-elements of the `Optimiser` element. The following elements are needed for both optimization types.

- `OptimiserType` Mandatory. Values are 'PGA', 'PGAPSO', or 'DirectSearch'. This defines the type of optimization to run. PGA uses a parallelized genetic algorithm, PGAPSO uses a particle swarm based approach, and DirectSearch uses simulated annealing (not parallelized).

- `Popsize` Mandatory. The size of the population of parameter sets. Values must be positive integers. Increasing the population will increase the coverage of parameter space, but will take longer to run. This should be an even number, >=4, if PGA is chosen, in order for the genetic algorithm to work properly. For 'DirectSearch', Popsize should be set to 1.

- `NumGen` Mandatory. A positive integer indicating the number of generations to permute the parameter sets.

  numGen x (popSize - numBest) = number of cost function evaluations or one iteration of loop.

  Default: 50 which is suitable for a quick initial test.

  If `NumGen`==`MaxNumGen`, no looping will occur.

- `MaxNumGen` Mandatory. A positive integer indicating the absolute upper limit on the cumulative total of generations run. SBSI-Numeric will terminate after this limit is reached, if no other stop criteria are met. This value, in conjunction with NumGen, will determine the looping behaviour of the framework. MaxNumGen / NumGen is the number of loops that will run. So, if `NumGen==MaxNumGen`, optimization will not loop, but will run in one invocation.

- `MaxNoChange` Mandatory. A positive integer describing the number of generations over which the improvement in cost function will be considered. If the cost function value does not improve over this period SBSI-Numeric will terminate. A default value is that `NumGen==MaxNumGen`, in which case this element has no effect.

- `MaxSimilarity`Mandatory, an integer percentage value. Optimization will stop if this percentage of parameter sets in the population are identical. The default value is 95%.

- `RestartFreq` Mandatory, a positive integer indicating the number of generations after which optimization will restart with a fresh parameter set if there is no improvement in cost function.

- `TargetCostFunctionValue` Mandatory. A positive double defining the cost function at which optimization will terminate.

These elements configure the output of the optimization:

- `Verbose` Mandatory, an integer. If `Verbose` > 0 it defines the interval of generations at which the population will be dumped into the PGALog file. If value == 0 then no output is generated. Default : -1 (this has the same effect as a value of 1 - i.e., every generation).

- `PrintFreq` Mandatory, a positive integer indicating the number of generations between outputs of the current best cost value Default =1. (This means that every generation will be output)

- `ResultDir` An optional element whose value is a file path to a folder where results will be stored.

- `CkpointDir` An optional element whose value is a file path to a folder where checkpoint information will be stored.

By default, results and checkpoints are stored within the folder chosen in which to run the optimization job.

The following elements are unique to PGA:

- `NumBest` Mandatory. A positive integer defining the number of parameter sets to proceed to the next generation. Default =2.

- `Seed` Mandatory. This is the seed for random number generation if PGA is chosen. If Seed <= 0 then the system will use its own seed randomly every time. Default = -1. This means the system will use a new seed each run. To ensure constant results, use seed > 0.

- `MutantProbability` Mandatory. The probability that a parameter will be mutated each generation. This should be a value between 0 and 1, which represents the proportion of parameters to be mutated from one generation to the next of the PGA. E.g., for a model with 100 parameters, if mutProb=0.1, then 10 parameters will be mutated. A mutant probability of 1 means that all parameters will be mutated. Mutation can cause the parameter to have any value within its constraints. A default setting is -1, where 1 parameter will be mutated each generation. The maximum value is 1 (but this is not recommended).

The following elements are unique to DirectSearch:

- `Mu` Mandatory. This controls the behaviour of simulated annealing. The greater the value of mu, the lower the probability of accepting parameters with higher cost function values. Default : 0.7

- `SobolSeed` optional, if user has a specific idea for the sobol seed. This is only needed for multiple loops, and SBSI-Numeric generates a default value. Each loop, the SobolSeed value is retained to avoid duplication of parameter space search. In most cases the value generated by the framework will be suitable.

The following elements are unique to PGAPSO:

- `PSONumGen` Mandatory if the optimizer type is PGAPSO. This parameter determines the number of generations to run a particle swarm algorithm for. If `PSONumGen` < `NumGen`, a hybrid approach is used, where a particle swarm procedure runs for `PSONumGen` generations, followed by a genetic algorithm procedure until `NumGen` is reached. If `PSONumGen` > `NumGen`, a pure particle swarm algorithm is used. In this case, 'looping' does not occur and the algorithm terminates after `PSONumGen` generations. For example, consider the configuration:

  `PSONumGen`=70, `NumGen`=100, `MaxNumGen`=400

  In this case, the program will run particle swarm for 70 generations, genetic algorithm for 30 generations, and then repeat this four times until `MaxNumGen` is reached (or a termination criterion is satisfied).

  In the following configuration:

  `PSONumGen`=200, `NumGen`=100, `MaxNumGen`=400

  the program will run particle swarm for 200 generations and then terminate. No genetic algorithm is run, as `PSONumGen` > `NumGen`.

6.7 Cost function configuration

Cost functions can be configured at two stages of optimization: setup, and optimization proper. Both `Setup` and `Optimizer` elements must contain at least one `ModelCostFunction` element, such as the one below.

```
<ModelCostFunction>
   <Type>X2Cost</Type>
   <Weight>0.5</Weight>
   <DataSetConfig>
      <FileName>ABC\_dat.dat</FileName>
      <DataFileStartTime>0.05</DataFileStartTime>
      <Interval>0.001</Interval>
      <ScaleType>None</ScaleType>
      <ValueType>Direct</ValueType>
   </DataSetConfig>
</ModelCostFunction>
```

These are described more full here:

- `Type` - mandatory: must be one of 'FFT' or 'X2Cost'. FFT should only be used when oscillations are sought for a particular species
- `Weight` - optional: Contains a float which the overal value of the cost function should be multiplied by. This allows weighting between separate model cost functions when there are more than one of them. When two or more cost functions are used, they can produce wildly different values, for example the X2Cost function may be evaluating to numbers of the order of 1.0, whilst the FFT cost function is evaluating to numbers of the order of 0.0001. This means that the FFT cost function will not direct the search because any change in this is consumed by the change in the X2Cost. Hence in this situation one may either weight the FFT function heavily by giving it a weight of 10000, or alternatively weight the X2Cost function less by giving it a very small weight such as 0.0001. The target cost function value though should be adjusted to account for this.
- `DataSetConfig` At least one element of this type is needed. The contents of each `DataSetConfig` element differ depending on whether FFT or X2 cost functions have been chosen.

6.7.1 Chi-squared cost configuration

- `FileName` The path to an experimental data file, in SBSI data format. This can be a relative or absolute path. If running SBSI-Numeric through loop.sh, the file can be in the job folder.

- `DataFileStartTime` The time relative to t0 in the model time series simulation, at which fitting should start. So for example if the model starts at t = 0 and has an entrainment phase for 32 hours, and measurement starts after this, then DataFileStartTime would be 32h, corresponding to the start of measured data.

- `Interval` This configures the granularity of the results, and should be the same granularity as the experimental data. For example if data set has time points 2.01, 2.02, 2.03 etc., then `Interval` should be 0.01. This also configures the default interval of the time series output, however see the optional 'ReportInterval' element of the solver configuration **??**.

- `ScaleType` An enumeration of `NONE` (uses raw data), `AbsAverage`, `Average`, `MaxMin`, `Fix` (not available yet). These options describe how to adjust the raw time-series output from a simulation before the simulation results are compared to the experimental results by the Chi-squared cost function. Note, that the same function is *not* applied to the data provided, these are functions that are expected to have been performed on the data prior to input. In other words, these allow the output of numerical analysis to be adjusted in the same way that we expect the experimental data to have been.

  - `NONE` The default, performs no transformation of the simulation output.
  - `Average` Each data point in a time series for a variable is divided by the average value for that variable over the time series.
  - `AbsAverage` Each data point in a time series for a variable is divided by the average of the absolute value for that variable over the time series.
  - `MaxMin` Each data point in a time series for a variable is normalized to a value between 0 and 1 using the equation $val - min(val)/max(val) - min(val)$ where max is the maximum value for the variable over the time series, and min the minimum value. This can be used if the data values are already scaled between 0 and 1.

  After the optimization finishes, the result folder includes untransformed output time series using the best parameters found during optimization. It may be useful (when visually comparing the simulation to the experimental data) to transform this simulation data using the same ScaleType as was used during the optimization. The **Normalise** utility in the util/ folder of SBSINumerics performs this function. To run use the following command line :

  ```
  Normalise  DataFile scaletype
  ```

  where 'scaletype' is one of the above options.

- `ValueType` One of `Direct` or `Normed`.

This influences the cost function values returned by the Chi-squared cost function. Using option `Direct` does not modify the calculated cost function value at all. However, this can be a problem. Imagine we have experimental data sets with the values at different scales:

Species X			Species Y		
Sim	Exp	Raw Cost	Sim	Exp	Raw Cost
0.15	0.1	$0.05^2 = 0.0025$	15	10	$5^2 = 25$
0.25	0.2	$0.05^2 = 0.0025$	25	20	$5^2 = 25$
0.35	0.3	$0.05^2 = 0.0025$	35	30	$5^2 = 25$
0.45	0.4	$0.05^2 = 0.0025$	45	40	$5^2 = 25$
		Sum of costs = **1**			Sum of costs = **100**

Clearly, when summing all the costs, using the `Direct` option, the cost of species Y will dominate the fitness function. If we want to make sure that all the data curves contribute equally to the cost, we can use the `Normed` option. When `Normed` is chosen, each data point value is divided by the square of the average of all the data points for that variable. E.g.,

$$NormedCost_i = Cost_i * \frac{1}{(\frac{\sum_{i=1}^{n}}{n})^2} \tag{6.1}$$

where i is the i$^{\text{th}}$ data point in a time series, and n is the number of data points in a time series. This will result in the costs being as follows:

Species X			Species Y		
Sim	Exp	Normed Cost	Sim	Exp	Normed Cost
0.15	0.1	$0.05/0.25^2 = 0.04$	15	10	$25/25^2 = 0.04$
0.25	0.2	$0.05/0.25^2 = 0.04$	25	20	$25/25^2 = 0.04$
0.35	0.3	$0.05/0.25^2 = 0.04$	35	30	$25/25^2 = 0.04$
0.45	0.4	$0.05/0.25^2 = 0.04$	45	40	$25/25^2 = 0.04$
	Average= 0.25	Sum of costs = **0.16**		Average=25	Sum of costs = **0.16**

In this case, where the ratio of experiment value : simulation value is the same for both species, the costs are now equal.

6.7.2 Configuration of FFT cost function

For configuration of an FFT cost function, the following structure of a DataSetConfig is used.

```
<DataSetConfig>
   <FileName>Exact.sbsiheader</FileName>
   <Interval>0.1</Interval>
   <State>
      <TargetName>Fn</TargetName>
      <TargetPeriod>24</TargetPeriod>
```

```
      <StartSampling>200</StartSampling>
      <EndSampling>400</EndSampling >
      <FncAmplitude>None</FncAmplitude>
   </State>
</DataSetConfig>
```

A description of the elements is as follows:

- `FileName` - header file (.sbsiheader or .hdr?) - a header file with parameters or initial values overriding the model.
- `Interval` - see the description for the chi-squared validation.
- `TargetName` - the ID of the species expected to oscillate.
- `Target period` - the expected period of oscillation for the target species, in the same time units as the model
- `StartSampling` - The time-point in the model simulation to start searching for oscillations.
- `EndSampling` - The time-point in the model simulation to stop searching for oscillations. This is an optional element, by default the function will search until the simulation endpoint.
- `FncAmplitude` - This is the function by which the FFT cost for the given state is factorised with respect to the amplitude. There are three possible values, None, Funcational and Max. If "None" is specified the FFT cost is not scaled with respect to the amplitude of the peak and is therefore amplitude agnostic. If this value is set to "Functional" (or its synonym "ReciprocalSqrtAmp") then the FFT cost is scaled by $1/\sqrt{amp}$ where amp is the amplitude of the peak. If this value is set to "Max" (or its synonym "ReciprocalSqrtLogAmp") then the FFT cost is scaled by $1/\sqrt{\log(amp)}$.

6.8 Solver configuration

Configuration of the solver is performed in the `Solver` element, for example:

```
<Solver>
  <TFinal>-1</TFinal>
  <TInit>0</TInit>
  <Interval>0.0001</Interval>
  <ReportInterval>0.5</ReportInterval>
  <MaxTimes>100000</MaxTimes>
  <Atol>1E-14</Atol>
```

```
  <Reltol>0.0001</Reltol>
  <SolverType>CVODESolver</SolverType>
</Solver>
```

A description of the elements is as follows:

- `SolverType` An enumeration of solver types. Currently this has one allowed value: 'CVODESolver' as this is the only solver type available.
- `TFinal` The end time point of model simulation. By default, this will be set to the latest time point of all datasets. There are several situations to consider, to ensure that the simulation time period is set up to work sensibly with the time period of the experimental data:

  If using the chi-squared cost function, there are various restrictions on TFinal:

  - A single data set is used. TFinal can be any point satisfying the condition

    $$0 < TInit < TFinal <= dataSetendpoint. \tag{6.2}$$

    So, for example, if the 'tail' of the data is needed to be excluded from fitting, then it can be.
  - Multiple data sets are used, covering the same time interval. In this case, the restrictions on TFinal are as described in case a) above.
  - Multiple data sets are used, covering different time intervals. For example dataSet1 covers time 0-100, dataSet2 covers 100-200. In this case, TFinal must be late enough to overlap with the 'latest' dataset (dataSet2) else the cost function will fail, trying to compare experimental data to a non-existent simulation. So in this case `TFinal` would need to be > 100.

  If using the FFT cost function: In order to give the cost function as good a chance as possible to find oscillations, `TFinal` - `TInit` MUST be greater than the longest `TargetPeriod`, and ideally SHOULD be greater than twice the `TargetPeriod`. The simulation time must overlap with the `StartSampling` time as well. For example, given the following FFT configuration:

```
<State>
    <TargetName>Fn</TargetName>
    <TargetPeriod>24</TargetPeriod>
    <StartSampling>200</StartSampling>
</State>
```

  then `TFinal` should be at least 248 (assuming `TInit` == 0). I.e., 200 + 2 * 24 and `TFinal` MUST be >= 224 (i.e., 200 + 24).

- `TInit`

  The Initial time point of model simulation, this should be 0 and in most circumstances will not need to be changed — this is the 'absolute zero' time point on which other times are based. Default — 0

- `MaxTimes`

  Maximum number of time steps in the simulation. This should satisfy the equation:

  $TInit + maxTimes * interval >= TFinal$

  Default — 100000

- `Interval` (Advanced user)

  Time distance between points in time-series solution. This should, at a minimum, be at least as often as the granularity of experimental data — e.g., if experimental data is at 0, 0.5, 1 etc, `Interval` =0.5 is a maximum interval. Default — 0.001

- `ReportInterval` An optional element to specify the time interval reported in results time series files. Similar to the above 'Interval' except that this affects how the best time series are reported to the user. The best timeseries are saved to file for user consumption (and/or plotting). Sometimes the 'Interval' must be set to a small value in order to allow the cost function to accurately match simulated data points to experimental data points. However this can lead to large time series files. Hence the 'ReportInterval' element allows the user to restrict this by setting a 'ReportInterval' larger than 'Interval'. This value will be treated as a multiple of 'Interval' in that when printing out the timeseries we will only print out every 'ith' line, where $i = ReportInterval/Interval$. Note however, that 'Interval' itself may be overridden by the model cost function setup. The 'Interval' element in the 'Solver' configuration specifies a step size for the numerical solver to use, while the 'Interval' within the 'ModelCostFunction' element specifies how many time points should be recorded and used in comparison against experimental data. Finally then the 'ReportInterval' will be the 'ReportInterval' used when writing the timeseries to a file.

- `Atol` (Advanced user)

  Absolute tolerance of the solver, recommended by Sundials CVODE solver. It is unlikely this will need to be altered in standard usage. Default — 1.0e-14

- `Reltol` (Advanced user)

  Relative tolerance of the solver. It is unlikely this will need to be altered in standard usage. Default — 1.0e-04

- Advice on setting Atol and Reltol These options are passed straight to the CVodes server of the Sundials library. Hence the Cvodes library documentation available

at: `https://computation.llnl.gov/casc/sundials/main.html` should be consulted for information. However we include verbatim from the cvodes manual some advice on setting the absolute and relative tolerances:

> For many users, the appropriate choices for tolerance values in reltol and abstol are a concern. The following pieces of advice are relevant. (1) The scalar relative tolerance reltol is to be set to control relative errors. So reltol = 104 means that errors are controlled to .01%. We do not recommend using reltol larger than 103 . On the other hand, reltol should not be so small that it is comparable to the unit roundoff of the machine arithmetic (generally around 1.0E-15). (2) The absolute tolerances abstol (whether scalar or vector) need to be set to control absolute errors when any components of the solution vector y may be so small that pure relative error control is meaningless. For example, if y[i] starts at some nonzero value, but in time decays to zero, then pure relative error control on y[i] makes no sense (and is overly costly) after y[i] is below some noise level. Then abstol (if scalar) or abstol[i] (if a vector) needs to be set to that noise level. If the different components have different noise levels, then abstol should be a vector. See the example cvRoberts dns in the cvode package, and the discussion of it in the cvode Examples document [15]. In that problem, the three components vary betwen 0 and 1, and have different noise levels; hence the abstol vector. It is impossible to give any general advice on abstol values, because the appropriate noise levels are completely problem-dependent. The user or modeler hopefully has some idea as to what those noise levels are. (3) Finally, it is important to pick all the tolerance values conservately, because they control the error committed on each individual time step. The final (global) errors are some sort of accumulation of those per-step errors. A good rule of thumb is to reduce the tolerances by a factor of .01 from the actual desired limits on errors. So if you want .01% accuracy (globally), a good choice is reltol = 106 . But in any case, it is a good idea to do a few experiments with the tolerances to see how the computed solution values vary as tolerances are reduced.

As a final warning, the graphs in Figure 6.1 show how much effect the setting of the 'RelTol' value can have on the numerical solution to a given model.

6.9 Worked examples

In this section we will give a few complete configuration examples to illustrate concepts described above

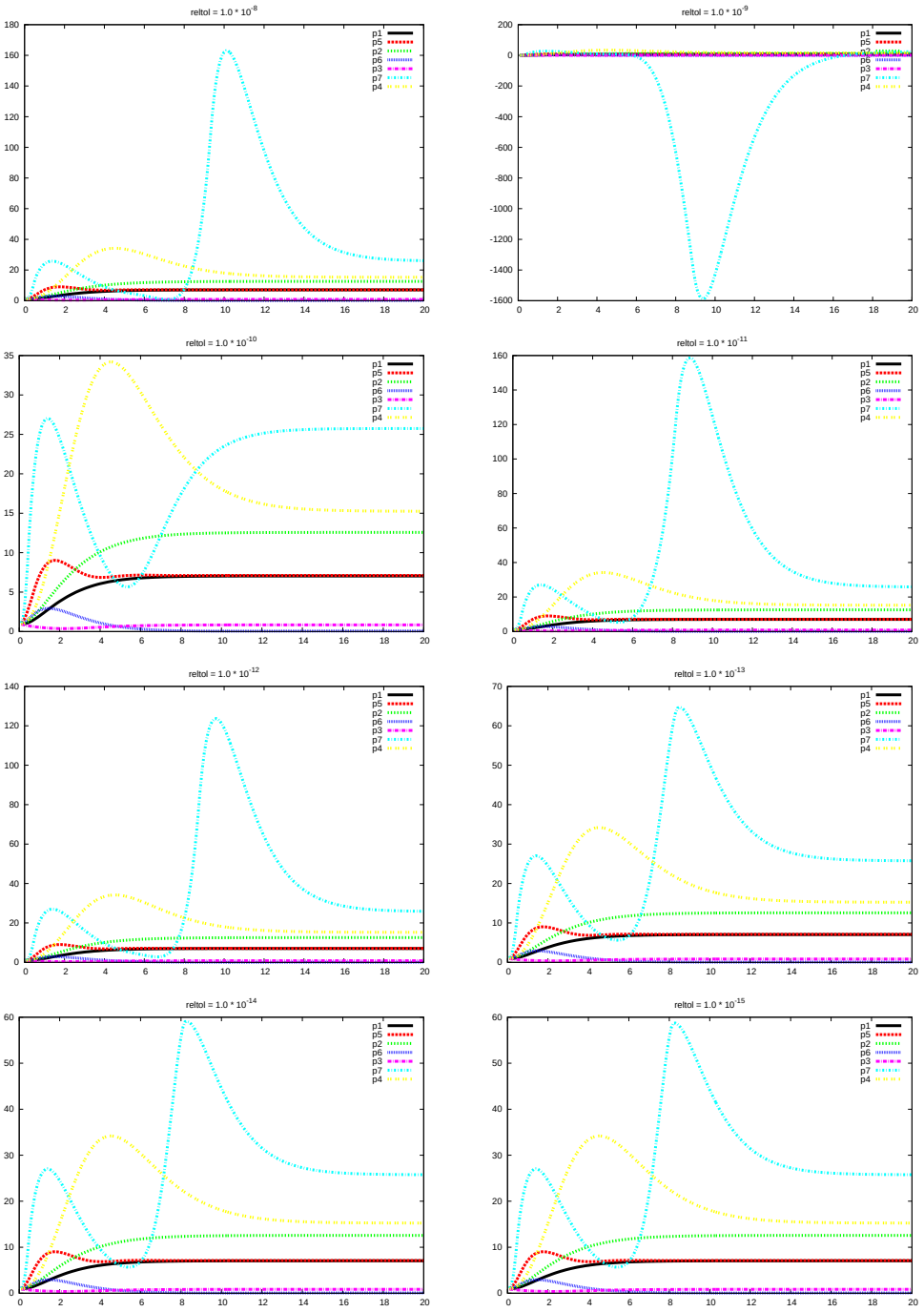

Figure 6.1: These graphs are all produced from the same model with the same parameters. The only difference between each solve, was the setting of the reltol configuration value. The consistency of the final three graphs convince us that we have found an appropriately small value for reltol.

6.9.1 The simplest ABC model

Here is the complete configuration file for the example abc_1 model:

```
<SBSINumericsConfig>
   <Setup>
<MaxCost>1000</MaxCost>
     <MaxRandomNumberGenerator>5000</MaxRandomNumberGenerator>
     <SetupType>ReadinFile</SetupType>
     <CkpointNum>50</CkpointNum>
     <ParameterInitialFile>abc_1_InitParam.dat</ParameterInitialFile>
     <ModelCostFunction>
<Type>X2Cost</Type>
<DataSetConfig>
<FileName>ABC_dat.dat</FileName>
<DataFileStartTime>0.05</DataFileStartTime>
<Interval>0.0001</Interval>
<ScaleType>None</ScaleType>
<ValueType>Normed</ValueType>
</DataSetConfig>
</ModelCostFunction>
   </Setup>
  <Optimiser>
     <OptimiserType>PGA</OptimiserType>
    <NumGen>50</NumGen>
    <MaxNumGen>100</MaxNumGen>
    <MaxNoChange>100</MaxNoChange>
    <MaxSimilarity>95.0</MaxSimilarity>
    <RestartFreq>51</RestartFreq>
    <PrintFreq>2</PrintFreq>
    <TargetCostFunctionValue>0.001</TargetCostFunctionValue>
    <CkpointFreq>-1</CkpointFreq>
    <ParameterInitialFile>abc_1_InitParam.dat</ParameterInitialFile>
    <PopSize>6</PopSize>
    <NumBest>2</NumBest>
    <MutantProbability>-1</MutantProbability>
    <Verbose>-1</Verbose>
    <Seed>-1</Seed>

    <ModelCostFunction>
      <Type>X2Cost</Type>
      <DataSetConfig>
        <FileName>ABC_dat.dat</FileName>
        <Interval>0.001</Interval>
```

```
        <DataFileStartTime>0.05</DataFileStartTime>
        <ScaleType>None</ScaleType>
        <ValueType>Direct</ValueType>
      </DataSetConfig>
    </ModelCostFunction>
  </Optimiser>
  <Solver>
    <TFinal>-1</TFinal>
    <TInit>0.5</TInit>
    <Interval>0.0001</Interval>
    <MaxTimes>100000</MaxTimes>
    <Atol>1E-14</Atol>
    <Reltol>0.0001</Reltol>
    <SolverType>CVODE_Solver</SolverType>
  </Solver>
</SBSINumericsConfig>
```

Let's look at this section by section:

Setup

```
<Setup>
 <MaxCost>1000</MaxCost>
 <MaxRandomNumberGenerator>5000</MaxRandomNumberGenerator>
 <SetupType>SobolSelect</SetupType>
    <ParameterInitialFile>abc_1_InitParam.dat</ParameterInitialFile>
   <ModelCostFunction>
   <Type>X2Cost</Type>
      <DataSetConfig>
       <FileName>ABC_dat.dat</FileName>
       <DataFileStartTime>0</DataFileStartTime>
       <Interval>0.001</Interval>
       <ScaleType>None</ScaleType>
       <ValueType>Normed</ValueType>
      </DataSetConfig>
   </ModelCostFunction>
</Setup>
```

This setup will attempt attempt to generate parameter sets using the Sobol random number generator (`SobolSelect`). These random parameter values will lie within the constraints specified in the file *abc_1_InitParam.dat*. For each candidate parameter set generated, Setup will evaluate the cost compared with data in the file ABC_dat.dat using the Chi-squared cost function. If cost < 1000, that parameter set will be entered into the starting population (whose size is specified by `PopSize` in the `Optimizer` element).

Set up will continue generating candidate parameter sets until either the population is complete, or it has tried more than 5000 parameter combinations.

For the cost function, the scale type is NONE- this means that the experimental data is on the same scale as the values generated by the simulation. Cost function values are normalized, and the data set time start is not offset in any way from the simulation start time (`DataFileStartTime` == 0).

The output of the setup can be seen in the *output_0.log* file.

```
  ../src/Setup.cpp runIn Setup: 0th cost is 13.4363. Generated 0/6 of starting populatio
../src/Setup.cpp runIn Setup: 1th cost is 13.8236. Generated 1/6 of starting population.
../src/Setup.cpp runIn Setup: 2th cost is 15.9547. Generated 2/6 of starting population.
../src/Setup.cpp runIn Setup: 3th cost is 13.715. Generated 3/6 of starting population.
../src/Setup.cpp runIn Setup: 4th cost is 12.5214. Generated 4/6 of starting population.
../src/Setup.cpp runIn Setup: 5th cost is 14.0815. Generated 5/6 of starting population.
../src/Setup.cpp runIn Setup: 6th cost is 13.8017. Generated 6/6 of starting population.
```

In this case, because of the simplicity of the model, all attempts to generate an initial parameter set have been successful, and the starting population of 6 has been generated in 6 attempts.

Optimization

```
 <Optimiser>
     <OptimiserType>PGA</OptimiserType>
    <NumGen>50</NumGen>
    <MaxNumGen>100</MaxNumGen>
    <MaxNoChange>100</MaxNoChange>
    <MaxSimilarity>95.0</MaxSimilarity>
    <RestartFreq>51</RestartFreq>
    <PrintFreq>2</PrintFreq>
    <TargetCostFunctionValue>0.001</TargetCostFunctionValue>
    <CkpointFreq>-1</CkpointFreq>
    <ParameterInitialFile>abc_1_InitParam.dat</ParameterInitialFile>
    <PopSize>6</PopSize>
    <NumBest>2</NumBest>
    <MutantProbability>-1</MutantProbability>
    <Verbose>-1</Verbose>
    <Seed>-1</Seed>

    <ModelCostFunction>
      <Type>X2Cost</Type>
      <DataSetConfig>
        <FileName>ABC_dat.dat</FileName>
        <Interval>0.001</Interval>
```

```
        <DataFileStartTime>0</DataFileStartTime>
        <ScaleType>None</ScaleType>
        <ValueType>Direct</ValueType>
      </DataSetConfig>
    </ModelCostFunction>
  </Optimiser>
```

In this configuration we have chosen a parallel genetic algorithm (`OptimiserType`=PGA) to run for a maximum of 100 generations (`MaxNumGen`=100) with a population of 6 parameter sets (`PopSize` = 6). We have chosen values of `RestartFreq` & `MaxNoChange` not to apply in this case, as both values are greater than `NumGen`. Result files will be generated at generations 50 and 100 (multiples of `NumGen`), or if the target cost function value of 0.001 is reached. The verbosity of logging is quite high (`Verbose`=1 means parameter values are output to PGA.log at every generation, and `PrintFreq`==2 means that the contents of the output.Log files describe the cost every two generations.) Parameters whose value is -1 are left to the system to choose an appropriate default value. In the case of `Seed`, this will generate different results on each invocation.

So, in the output.Log file, we will see some output as follows:

```
      ++++++++++++++++++++++
PGASOBOL::Initialise.
Starting PGSWrapper::Run
PGARun
PGARunGM
Iter #     Field      Value
1          Best       8.921648e-01
2          Best       6.129037e-01
4          Best       5.282705e-01
6          Best       5.159056e-01
8          Best       3.874666e-01
10         Best       3.874666e-01
12         Best       3.370939e-01
14         Best       2.994815e-01
16         Best       2.562934e-01
```

In this output, column 0 ('Iter')tells you the generation time, and 'Value' shows the cost function value at that generation).

More verbose information can be found in the PGALogs. Finally, assuming that the target cost function is not reached, there will be output time series generated at generations 50 and 100, These are generated in SBSI data format.

6.9.2 An example with FFT cost function configuration

In this example, we configure a small Circadian clock model derived from Goldbeter's Neurospora clock model. A complete configuration file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<OFWConfig>
<Solver>
  <SolverType>CVODESolver</SolverType> <!-- the only choice so far -->
<TFinal>-1</TFinal>
<TInit>0</TInit>
          <Interval>0.01</Interval>
<MaxTimes>1000000</MaxTimes>
<Atol>1.0e-14</Atol>
<Reltol>1.0e-04</Reltol>
</Solver>
<Setup>
<SetupType>SobolSelect</SetupType>
<MaxCost>50</MaxCost>
<MaxRandomNumberGenerator>1000000</MaxRandomNumberGenerator><!--MAXRG-->
<ParameterInitialFile>InitParam.dat</ParameterInitialFile>
<ModelCostFunction>
 <Type>FFT</Type>
<DataSetConfig>
<FileName>Exact.sbsiheader</FileName>
<Interval>1.0</Interval>
<State>
<TargetName>mRNA</TargetName>
<TargetPeriod>20</TargetPeriod>
<StartSampling>150</StartSampling>
<EndSampling>250</EndSampling>
</State>
<State>
<TargetName>Fc</TargetName>
<TargetPeriod>20</TargetPeriod>
<StartSampling>150</StartSampling>
<EndSampling>250</EndSampling>
</State>
<State>
<TargetName>Fn</TargetName>
<TargetPeriod>20</TargetPeriod>
<StartSampling>150</StartSampling>
<EndSampling>250</EndSampling>
</State>
```

```
</DataSetConfig>
</ModelCostFunction>
                <ModelCostFunction>
                         <Type>X2Cost</Type>
                        <DataSetConfig>
                              <FileName>Exact_5.dat</FileName>
                              <DataFileStartTime>0</DataFileStartTime>
                              <Interval>5.0</Interval>
                              <ScaleType>None</ScaleType>
                              <ValueType>Normed</ValueType>
                              </DataSetConfig>
                </ModelCostFunction>
</Setup>
<Optimiser>
<OptimiserType>PGA</OptimiserType>
<PopSize>64</PopSize>
<NumGen>200</NumGen>
<MaxNumGen>400</MaxNumGen>
<NumBest>2</NumBest>
<MaxNoChange>60</MaxNoChange>
<MaxSimilarity>95</MaxSimilarity>
<RestartFreq>80</RestartFreq>
<Seed>1213</Seed><!-- unique to PGA -->
<PrintFreq>10</PrintFreq>
<Verbose>0</Verbose>
<TargetCostFunctionValue>1.0e-8</TargetCostFunctionValue>
<ResultDir>UserModel/results</ResultDir>
<CkpointDir>UserModel/ckpoint</CkpointDir>
<CkpointFreq>50</CkpointFreq>
<MutantProbability>-1</MutantProbability>
</Optimiser>
<Results>
<TimeSeries>
<TimeSeriesResultDir>UserModel/results</TimeSeriesResultDir>
<TimeSeriesFileName>Best_Tseries</TimeSeriesFileName>
</TimeSeries>
</Results>
</OFWConfig>
```

The main difference between this example and the ABC_1 example is that we are using the FFT (Fast Fourier Transform) cost function as well as Chi-squared. This cost function looks for oscillations in the model and returns a cost depending on how well the oscillations match the expected Period. Currently the costs are simply summed to get a global cost function value.

In the model, we have 3 species (Fc, Fn and mRNA) that we expect to oscillate with a 20 hour rhythm. We expect oscillations from t=150 onwards (to allow time for the oscillations to stabilise) and so it would be important to set T`Final` to be long enough to get at least two full oscillations in the model. In this case, assuming that oscillations begin at t=150, 5 oscillations with a 20h period could occur before the simulation endpoint at `Tfinal`=250.

In this configuration, we have also specified an optional `Results` element in which the default location and name of the output time series can be overridden.

During the optimization, this time optimization will stop if the cost function does not improve after 60 generations. This can be useful with larger models, in order to avoid wasting run-time on expensive supercomputers.

6.9.3 An example using simulated annealing (SA)

This section shows the configuration elements needed for simulated annealing:

```
<Optimiser>
  <OptimiserType>DirectSearch</OptimiserType>
    <PopSize>1</PopSize>
    <NumGen>100</NumGen>
    <MaxNoChange>60</MaxNoChange>
    <RestartFreq>80</RestartFreq>
    <SobolSeed>-1</SobolSeed>
  <PrintFreq>20</PrintFreq>
  <TargetCostfunctionvalue>0.001</TargetCostfunctionvalue>
  <CkpointFreq>50</CkpointFreq>
  <!-- two unique values for SA -->
  <Mu>10</Mu>
  <SobolSeed>-1</SobolSeed>
</Optimiser>
```

In this case, `PopSize` must be 1, and `Mu` is the main controlling parameter. In most cases `SobolSeed` can remain unchanged, using the default of -1.

Chapter 7

Examples for Phase Shifting

7.1 Introduction

This chapter is a tutorial on using the SBSI frameworks to optimise a model in which deterministic discrete events occur. Examples include the modelling of experiment conditions such as light or temperature entrainment.

There are two ways in which we may model a change in the model during the time course of the numerical analysis. The first uses explicit events whilst the second uses our ability to define functions which operate over the simulated time.

We begin with the trivial example model, which has a single reaction at a simple rate, the full SBML source of this model is shown in Figure 7.1.

```
k1 = 0.1
A -> B at rate A * k1
```

Our aim is to simulate three phases each of 20 time units to a final time of 60. In the first phase we wish for our parameter $k1$ to be equal to 0.1, in the second this switches to 0.001 and in the final phase we switch the value of $k1$ back to its original value.

7.2 Using Explicit Events

Explicit events are supported in SBML, all events are defined within a `listOfEvents` element using the `event` tag. An `event` contains a `trigger` and a list of event assignments. For more information please see the SBML specification at: `www.sbml.org`.

Figure 7.2 depicts the additional SBML which must be added to original base model shown in Figure 7.1. Note that there are two events for the three phases because there are only two phase switches. The list of events elements should be the sub-element of the 'model' element. It is important to note that in the definition of any parameter altered by an event assignment 'constant' attribute must be explicitly set to 'false'. So our original definition for the parameter 'k1' becomes:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version3"
      level="2" version="3">
  <model id="singleReaction">
    <listOfCompartmentTypes>
      <compartmentType id="Compartment"/>
      <compartmentType id="Membrane"/>
    </listOfCompartmentTypes>
    <listOfCompartments>
      <compartment id="main" size="1.0"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="A" compartment="main"
                      substanceUnits="item"
                      hasOnlySubstanceUnits="true"/>
      <species id="B" compartment="main"
                      substanceUnits="item"
                      hasOnlySubstanceUnits="true"/>
    </listOfSpecies>
    <listOfParameters>
      <parameter id="k1" value="0.01"/>
        <parameter id="k2" value="0.001"/>
    </listOfParameters>
    <listOfInitialAssignments>
      <initialAssignment symbol="A">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <cn> 100.0 </cn>
        </math>
      </initialAssignment>
      <initialAssignment symbol="B">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <cn> 100.0 </cn>
        </math>
      </initialAssignment>
    </listOfInitialAssignments>
    <listOfReactions>
      <reaction id="a" reversible="false">
        <listOfReactants>
          <speciesReference species="A"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="B"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
              <times/>
              <ci>A</ci>
              <ci>k1</ci>
            </apply>
          </math>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```


Figure 7.1: The full SBML source for our simple example model

```
<listOfParameters>
  <parameter id="k1" value="0.01" constant="false" />
</listOfParameters>
```

However this represents the only *modification* to the model required in order to use explicit events. With this done the events may added or subtracted from the model without otherwise modifying the SBML model.

The results of evaluating this model are shown in Figure 7.3.

7.3 Using Time-Dependent Rate Functions

Often the model is not authored in SBML, but is authored in another modelling framework such as Bio-PEPA and the SBML automatically generated. In this situation it may be awkward, if the particular framework involved does not support SBML events, to generate the SBML and then manually add the required events. In this case it may be more convenient to use a time-dependent rate function.

In Bio-PEPA to achieve the same scenario as that above we would write the following rate function:

```
r = [ A * ( ( (H(20 - time) + H(time - 40)) * 0.1) +
            ( (H(time - 20) * H(40 - time)) * 0.001) )
    ];
```

To understand this we must understand first of all the H function in Bio-PEPA. This function returns 1 if the argument is greater than zero and 0 otherwise. So the first call to the H function, $H(20 - time)$ will be 1 if the current time is less than 20. We add this to the result of calling $H(time - 40)$ which checks that the current time is above 40. Hence the addition of these will be 1 if we are in the first or third phase. This is then multiplied by the rate at which we wish 'k1' to be, if we are in the first or third phases. The next two calls are $H(time - 20)$ and $H(40 - time)$ check that we are within the bounds of the second phase, since the first checks that the time is above 20 and the second that it is below 40. Since the results of these two calls are multiplied together the overall result will be 1 only if both of these hold true and we are in fact in the second phase. The 1 or 0 is then multiplied by the rate which we wish the 'k1' parameter to take during the second phase. These two rates are then multiplied together and the entire rate expression is equal to A multiplied by the desired value of 'k1' depending on which phase we are in.

For other formalisms we can write the helper SBML function definitions given in Figure7.4.

```
<listOfEvents>
  <event>
    <trigger>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <gt/>
          <csymbol encoding="text"
                   definitionURL="http://www.sbml.org/sbml/symbols/time"> t
          </csymbol>
          <cn>20.0</cn>
        </apply>
      </math>
    </trigger>
    <listOfEventAssignments>
      <eventAssignment variable="k1">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <cn>0.001</cn>
        </math>
        </eventAssignment>
      </listOfEventAssignments>
  </event>
  <event>
    <trigger>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <gt/>
          <csymbol encoding="text"
                   definitionURL="http://www.sbml.org/sbml/symbols/time"> t
          </csymbol>
          <cn>40.0</cn>
        </apply>
      </math>
    </trigger>
    <listOfEventAssignments>
      <eventAssignment variable="k1">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
          <cn>0.1</cn>
        </math>
        </eventAssignment>
      </listOfEventAssignments>
 </event>
</listOfEvents>
```

Figure 7.2: Events SBML which is added to the simple SBML model in order to invoke the desired changes in rates during simulation.

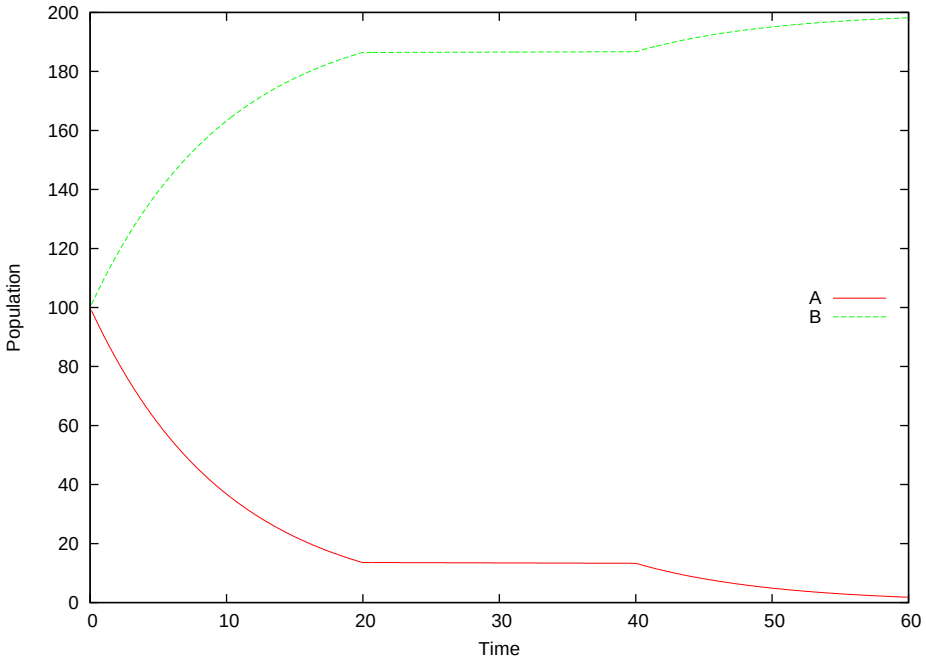

Figure 7.3: The results of numerical analysis over a model containing phase switches at times 20 and 40.

```
<functionDefinition id="phaseZeroToX">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <lambda>
    <bvar><ci>r</ci></bvar>
    <bvar><ci>X</ci></bvar>
    <bvar><ci>t</ci></bvar>
    <piecewise>
      <piece>
        <ci>r</ci>
        <apply>
          <gt/>
          <ci>X</ci>
          <ci>t</ci>
        </apply>
      </piece>
      <otherwise>
        <cn>0</cn>
      </otherwise>
    </piecewise>
  </lambda>
</math>
</functionDefinition>

<functionDefinition id="phaseXToY">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <lambda>
    <bvar><ci>r</ci></bvar>
    <bvar><ci>X</ci></bvar>
    <bvar><ci>Y</ci></bvar>
    <bvar><ci>t</ci></bvar>
    <piecewise>
      <piece>
        <piecewise>
          <piece>
            <ci>r</ci>
            <apply>
              <gt/>
              <ci>t</ci>
              <ci>X</ci>
            </apply>
          </piece>
          <otherwise>
            <cn>0</cn>
          </otherwise>
        </piecewise>
        <apply>
          <lt/>
          <ci>t</ci>
          <ci>Y</ci>
        </apply>
      </piece>
      <otherwise>
        <cn>0</cn>
      </otherwise>
    </piecewise>
  </lambda>
</math>
</functionDefinition>
```

```
<functionDefinition id="phaseXToInfinity">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <lambda>
    <bvar><ci>r</ci></bvar>
    <bvar><ci>X</ci></bvar>
    <bvar><ci>t</ci></bvar>
    <piecewise>
      <piece>
        <ci>r</ci>
        <apply>
          <gt/>
          <ci>t</ci>
          <ci>X</ci>
        </apply>
      </piece>
      <otherwise>
        <cn>0</cn>
      </otherwise>
    </piecewise>
  </lambda>
</math>
</functionDefinition>
```

Figure 7.4: . Helper functions for SBML time-dependent rate functions

Chapter 8

Glossary

This section describes some of the acronyms and terms used in the SBSI optimization framework.

OFW - Optimization framework

PGA - Parallel genetic algorithm

SA - simulated annealing

SBSI - Systems Biology Software Infrastructure

SBML - Systems Biology Markup Language

FFT - Fast Fourier Transform