StabilityChecker

Miriam Leon, Chris Barnes

July 27, 2015

Contents

Chapter 1

Introduction

Chapter 2

Installation

StabilityChecker has been developed to work on a Linux operating system, with GPU support.

The following dependencies are essential for the successful implementation of the package:

- numpy
- logging
- cuda-sim
- libsbml

```
$ cd StabilityChecker
$ python setup.py install
```

2.1 Linux

On linux this will copy the module `stabilitychecker` into the `lib/pythonXX/site-packages` directory corresponding to the python version that was used to invoke the installation.

The path then should be added by issuing

```
$ export PATH=<dir>:$PATH
```

in the shell `run.sh` script An `exe=<dir>` variable should also be added to the `run.sh` script pointing the script to the right package installation.

Chapter 3

Using the package

3.1 Conversion of SBML models to CUDA code

When `run-abc-sysbio` is run, model(s) written in SBML format are used to generate an appropriate code module representing the model, via a call to `abcsysbio_parser.ParseandWrite`. The format of the code written depends on the integration type, which also informs the program which solver to use to simulate the model. (See Section 3.3.)

In an SBML model, values can be assigned to compartment sizes, to parameters and to species. When the Python module representing the model is written, the model is inspected to determine how values should be assigned during the course of the simulation. Compartments typically have a constant size and are considered as an additional parameter to the model, albeit one that the user may not want to infer. Some species may be outside of the scope of the reactions in the model, having either a constant value or a value assigned by an assignment rule.

Conversely, SBML has the capacity for rate rules to be used to assign values to species, parameters or compartment sizes. Rate rules are time-dependent rules used to assign values to variables. Because our solvers typically assign values to species over time to compute a trajectory, species, parameters and compartment sizes with rate rules are treated as 'species' in a Python module representing the SBML model.

3.2 The user input file

Not all the information required for the algorithms is intrinsic to the model, therefore a .xml file that contains the additional information (such as initial conditions and parameter values) must be supplied to the script. This file, the "user input file", must be written in a specific format. Examples are included in the "Examples" folder which is supplied along with the package; details of the input file are given here.

3.2.1 Required arguments

modelnumber Number of the model for which details are described in this input file

epsilon This allows for the specification of the tolerance schedules. Often only one epsilon schedule is required but in more complex design problems or with inferences using summary statistics multiple epsilon schedules may be desired. Within the `epsilon` tags each vector of schedules can be specified via a whitespace delimited list of values between two tags eg `<e1> </e1>`. Note that the parser ignores the name of the tag and so will always read them in order. If multiple schedules are provided then the user must specify a custom distance function (see Chapter 3.3).

autoepsilon This is an experimental feature used instead of the **epsilon** option where the epsilon schedule is automated. The vector of final epsilons (whitespace separated) can be specified within `<finalepsilon> </finalepsilon>` tags. The `<alpha>` tag specifies the quantile of the previous population distance distribution to choose as the next epsilon. The optimal value of this parameter will depend on the models and data.

particles Number of particles to accept

beta Number of times to simulate each sampled parameter set. For deterministic systems beta is set to 1. For analysis of stochastic systems beta can be chosen larger than 1.

dt The internal time step for the solvers.

data This section is divided into `<times>` and `<variables>`. Both describe the experimental data for which the parameters or models respectively have to be analyzed.

times The time points are given within `<times>` tags as a whitespace delimited list.

variables The species concentrations (in the case of an ODE or SDE simulation) or molecule numbers (in the case of a Gillespie simulation) are also given as whitespace delimited lists and denoted as `<v1>`, `<v2>` .. `<vN>` for N different species. Note that the names of the tags is ignored and the parser always reads the species in order. Missing data can be specified provided that the first entry in present. Missing data is denoted by 'NA'.

models Each model is contained within tags `<modeli>` ($i = 1, \ldots, M$, where M is the total number of models to be investigated.)

name The name of the model which will be the name of the code written (with suitable extension .py, .cpp, .cu) to represent the model in a format that is interpretable by the solvers. The name of the code model file if option `--localcode` is given.

source The name of the .xml file containing an SBML representation of the model. Can be left blank if option `--localcode` is given.

type The simulation type. One of `ODE`, `SDE` or `Gillespie`.

fit Denotes the correspondence between the species defined in the SBML model and the experimental data. If this keyword is not given, or if **fit** is `None`, all the species in the model are fitted to the data in the order that they are listed in the model. Otherwise, a comma-delimited list of fitting instructions with the same length as the dimensions of the data can be supplied. Simple arithmetic operations can be performed on the species from the SBML model. To denote the `N`th species in the SBML model, use `speciesN`. For example, to fit the sum of the first two species in the model, write **fit**: `species1+species2`.

parameters, initial Prior specifications on parameters and initial conditions. Note that the tag names for each parameter and species initial condition are ignored and are always read in the order specified. The prior is specified within the tags via a whitespace delimited list:

- constant x: constant parameter with value x
- normal a b: lognormal distribution with location a and var b
- uniform a b: uniform distribution on the interval $[a, b]$
- lognormal a b: lognormal distribution with location a and var b

3.2.2 Output

The outputs from running the ABC SMC algorithm are saved in a folder specified via the `-of --outfolder` option.

- `_data.png`, a scatter plot of your input data.

- `rates.txt` containing population number, number of sampled particles, acceptance rate and time to complete in seconds
- `ModelDistribution_1.png` and `ModelDistribution.txt` Histograms of the posterior distribution of accepted models after each population. Above each histogram the population number, epsilon, and acceptance rate for that population are displayed.
- One text file per population, `distance_PopulationN.txt`, listing the distances of the accepted particles together with the model number of the accepted model.
- One text file per population, `traj_PopulationN.txt`, the trajectories of the accepted particles. Each line contains:
  accepted particle number, replicate number (==0 if beta=1), model id, fitted species id, X(t=1), X(t=2) .....
- One sub-folder per model. These sub-folders, suffixed with the model name, contain sub-folders for each population, `population_N`. Each contains:
  - `data_PopulationN.txt`, the accepted parameter sets
  - `data_WeightsN.txt`, the accepted parameter weights
  - `ScatterPlotPopulationN.png`, scatter plots of all accepted parameters. (See Figure **??**)
  - `TimeseriesPopulationN.png`, simulations of the model using ten accepted parameter sets, to compare with the data. Refer to Figure **??**.
  - `weightedHistograms_PopulationN.png`, histograms showing accepted parameter distributions.
- `copy` contains in binary data form the information required to restart the ABC SMC algorithm using the last population. These files are not human-readable but are read into Python if the algorithm is being run restarting from a previous population. See Example 2.

3.2.3 Editing the input file

To perform ABC SMC, some additional information does need to be added to the input file.

epsilons A whitespace delimited list of floats, these are the distances below which a parameter set is accepted by the algorithm.

particles Integer, the size of each population.

dt Float, the internal timestep. For a stiff model, a small **dt** is required for a successful simulation. However, the smaller the value of **dt**, the longer the simulation will take. The SIR models are not particularly stiff so **dt** can be set to 1.

Some model-specific information must be completed for each of the models.

name The name of the python file that will represent the model. Defaults to `model1`, `model2`, ... ,`modelN`. These have been renamed in the input file we are using.

source The name of the .xml file containing the SBML model. The source file(s) must be in the same working directory as the scripts and the input file.

type The integration type used to simulate the model. Defaults to `ODE`. Other integration types are `SDE` (see Example 2) or `Gillespie` (see Example 3).

fit A string, defaults to `None`. If **fit** is not `None`, it must take the form of a comma-delimited list of fitting instructions describing how to fit the species in the SBML model (denoted `species1, species2, ... , speciesN`) to the variables given in the **data**. Model 2 in this example describes the trajectories of four species, including the latent population L. However, our data only describes three variables. We give the fitting instruction

```
fit: species1, species3, species4
```

to denote that the first species in Model 2 describes variable 1 in the data, the third species in Model 2 describes variable 2 and the fourth species in Model 2 describes variable 3. Fitting instructions can include simple arithmetic operations. For example if data represent the sum of the first two species in a model the fitting instruction for that variable should be written `species1+species2`.

Priors For each parameter and initial condition we specify the distribution:

- constant value
- normal location variance
- uniform lower upper
- lognormal location variance

3.2.4 Running StabilityChecker

3.3 Extending the package

3.3.1 Defining your own clustering algorithms

Defining custom distance functions is straightforward. The user needs to to pass the argument