#Name : User Manual

Neil Dalchau, Andrew Phillips

July 30, 2015

Contents

1 Introduction

2 Fundamentals

2.1 Chemical reaction metworks (CRNs)

2.2 Domain-specific programming languages

2.3 Parameters

3 Simulation

A variety of simulation modes are available within the BME. These are described in detail in the forthcoming sections. All share a common format for simulation data and visualization, and also some common simulation options.

3.1 Common

3.1.1 Time-span

The time-span of a simulation can be specified using the `sample` directive, as follows:

```
directive sample FLOAT INT       // Arguments: end time, number of points
directive sample FLOAT,FLOAT INT // Arguments: start time, end time, number of points
```

3.2 Stochastic

3.3 Deterministic

3.4 Spatial

3.5 Chemical master equation (CME)

4 Inference

The inference module enables *parameter* values of a model to be inferred from observation data. Markov chain Monte Carlo (MCMC) is the methodology used, as implemented in the Filzbach software [**?**] (http://research.microsoft.com/filzbach). Filzbach uses a variation of the Metropolis-Hastings (MH) algorithm to perform Bayesian parameter inference. The MH algorithm is used to approximate the posterior probability of a parameter set from a hypothesised model taking on certain values, constrained by a likelihood function. The probability of each parameter value is then approximated by constructing a Markov chain of sampled parameter sets, such that a proposed parameter set is accepted with some probability, based on the ratio of the likelihood function evaluated at current and proposal parameter sets. For more information on MCMC methods, see [**?**]. MCMC methods, such as simulated annealing, have also been shown to efficiently find solutions to combinatorial optimisation problems [**?**], taking a stochastic search approach similar to the MH algorithm. Stochastic search can provide benefits over gradient-based optimisers by maintaining a nonzero probability of making up-hill moves, protecting against getting stuck in poor local optima.

4.1 The likelihood function and noise model

The likelihood function is formed by taking all data specified in the program, and comparing corresponding simulations in a probabilistic way. Each data-point contributes a term to the likelihood function. Each are assumed to be independent and Gaussian distributed, centred on the corresponding simulated value and with some variance σ^2. As there are potentially multiple datasets, each with potentially multiple input treatments and output species, there could be several time-courses being bundled into this single function. Nevertheless, their composition is straightforward. For each time-series with measured values η_k corresponding to times t_k, $k = 1, \ldots, n$, corresponding simulated values y_k are obtained. Then, the likelihood function is defined as

$$L(\theta|\text{data}) = \prod_{\text{datasets, inputs, outputs, } k} \left\{ P(\eta_k; y_k, \sigma^2) \right\} \tag{1}$$

A *noise model* can be applied to the variance σ^2. We currently support two such models. The default model is a *constant* noise model, in which the variance is independent of signal intensity, and is treated as an inference parameter. An alternative *proportional-error* model can also be specified, which assumes that $\sigma^2 = \alpha \times y_k$, with α treated as an inference parameter. Since each species being measured may have different noise characteristics, the noise parameters can be specialised to each species, or assumed to be homogenous (see below).

4.2 Specifiying inference options

Eventually, an MCMC algorithm will *converge*, and the samples of the chain will represent the joint posterior distribution $P(\theta|\text{data})$. As such, the more samples that are used, the more accurately will the samples numerically approximate this distribution. To prevent initial transients in the chain from skewing the joint posterior, early samples are often discarded. These samples are known as *burn-in* samples. Both the number of burn-in samples and post-burn-in samples can be specified using the `fit_run` directive. Filzbach also permits

```
directive fit_run { burnin = INT; samples = INT; burnin2 = INT; eststeps = INT; separatenoise =
    BOOL; noisemodel = INT }
```

Table 1: Inference options and their default values

Argument	Description	Default
`burnin`	Number of burn-in iterations, which are eventually discarded	1000
`samples`	Number of samples, which are stored	1000
`burnin2`	Number of burn-in iterations for MLE exploration	0
`eststeps`	Number of samples for MLE exploration	0
`thin`	Interval for thinning the posterior samples	10
`separatenoise`	Use a separate noise parameter for each species	`true`
`noisemodel`	Assumed distribution of error (0 – constant, 1 – proportional)	0

4.3 Parameters

For parameter inference, all parameters must be specified using the full syntax (cf. Section 2.3). The syntax for a single parameter is

```
directive parameters [ STRING, (FLOAT, FLOAT), FLOAT, ENUM, ENUM ]
```

where the arguments in order are the parameter name, the lower bound, the upper bound, a default value (for simulation), the scale over which to search (`real` or `log`), and finally a specification for the parameter being `fixed`, or to be inferred (as a single value - `random`, or as separate values for each input - `multiple`).

4.4 Datasets

Observational data can be loaded into the software from tab-delimited format. To be interpretable by the software, each data-file must conform to a set of standards. Each file must contain a single row of headers, which are simply used as labels for visual feedback (their contents do not change the program behaviour). The second column is reserved for specifying the time-points at which each row of data was collected. All subsequent columns cannot contain time values, and so if you have multiple time-courses with different time-points, they must be imported as separate files. The ordering of measurement columns must correspond with the ordering of the simulations specified in the program, with respect to different input treatments and output species. The ordering assumed groups the multiple outputs for each input, then horizontally concatenating these blocks. For example, a file reporting measurements of CFP and YFP at three different input concentrations would look like the example in Table 2.

Table 2: Example data file for inference.

Time	CFP (0 uM)	YFP (0 uM)	CFP (0.1 uM)	YFP (0.1 uM)	CFP (1 uM)	YFP (1 uM)
0.0	0.1	0.12	0.08	0.11	0.09	0.13
10.0	0.12	0.11	0.14	0.1	0.32	0.12
20.0	0.13	0.15	0.23	0.13	0.56	0.13
⋮	⋮	⋮	⋮	⋮	⋮	⋮

To load a data-file into the Visual tools, navigate to the ”Data Sets” sub-tab of ”Inference” on the right

half of the tool. The load button allows you to navigate to your data-file. Once loaded, a default name is given to the dataset, which can be modified by double-clicking on the entry in the "Name" column. The name appearing here is how the dataset is referenced in your model program.

4.5 Relating models to datasets

To specify how a model relates to each dataset you want to infer your parameters with, the `fit` directive is used. The argument to the fit directive is a tuple consisting of a named parameter sweep (see Section 2.3 that specifies the inputs, the name of a data-file, and a list of the measured species. Therefore, the example in Table 2 would be specified as

Should be changed to a record.

```
directive parameters [ c = 0.0 ]
directive sweep mysweep = { c = [0.0, 0.1, 1.0] }
directive fit { mysweep; mydatafile; [CFP; YFP] }
```

5 Verification

6 Summary of directives

Directive	Description	Explanation
`sample`	Time-span of a simulation	