1 Some guidance for developing new methods in BEAST 2

Disclaimer: below some ramblings on methods development for BEAST 2 (Bouckaert *et al.* 2014, Drummond and Bouckaert (2015), Bouckaert *et al.* (2019)) packages. This is a living document based on collected wisdom of BAEST developers. Use at own risk.

This document is about testing validity of a BEAST method, not the programming aspects (like setting up dependencies, wrapping up files into a package, etc.), which can be found in the tutorial for writing a BEAST 2 package and writing a package for a tree prior tutorial.

Levels of validation:

- the model appears to produce reasonable results on a data set of interest.
- the model produces more reasonable results on a data set of interest than other models.
- unit tests show correctness of direct simulator implementation, likelihood implementation and/or operator implementation(s).
- sampling from prior conforms to expectations.
- a simulation study shows parameters simulated under the model can be recovered by inference from simulated data for a fixed tree and fixed other parameters for a small number of illustrative cases.
- as previous but with sampled tree and sampled parameters, so the process is repeated N times and tree and parameters sampled from a reasonable prior.
- a simulation study shows the model can recover parameters (most of the time) even when there are model violations in simulating the parameters.

1.1 Testing new methods

New methods require usually require two parts: an implementation $I(M)$ of a model M and associated probability $p_I(\theta|M)$ of states θ, and MCMC operators $R(\theta) \rightarrow \theta'$ for creating proposals θ' for moving through state space starting in state θ (though sometimes just an operator is validated that is much more efficient than previously existing operators). This guide contains some procedures to make sure that the model and operators are correctly implemented. Ideally, we have an independent implementation of a simulator $S(M) \rightarrow \theta$ that allows (possibly inefficiently) to sample from the target distribution $p_S(\theta|M)$. If so, we also need to verify that the simulator is correctly implemented. In summary, we need to establish correctness of:

- the simulator implementation $S(M) \rightarrow \theta$ (if any)
- the model implementation $I(M)$
- operator implementations R

1.1.1 Verify correctness of simulator implementation

To verify correctness of a simulator implementation S for model M directly, the distributions $p_S(\theta|M)$ should match expected distribution based on theory. We can verify this by drawing a large number of samples using S, calculate summary statistics on the sample and compare these with analytical estimates for these statistics. For example, ...

When no analytical estimates of statistics are available, ...

Examples of simulators (this list is far from exhaustive):

- the MASTER (Vaughan and Drummond 2013) BEAST 2 package is a general purpose package for simulating stochastic population dynamics models which can be expressed in terms of a chemical master equation.
- SimSnap for SNAPP (Bryant *et al.* 2012) is a custom build implementation in C++ for simulating alignments for a fixed tree and SNAPP parameters.
- The `beast.app.seqgen.SequenceSimulator` class in BEAST 2 can be used to simulate alignments for general site models using reversible substitution models. See testSeqGen.xml for an example.
- Models implemented in other phylogenetic software packages, such a BEAS 1, MrBayes, RevBayes, allow sampling a distribution using MCMC.
- The `beast.core.DirectSimulator` class in BEAST 2 can be used to draw samples from distributions in BEAST that extend `beast.core.distribution.Distribution` and implement the `sample(state, random)` method. You can set up an XML file and run it in BEAST. Here are a few examples: testDirectSimulator.xml, testDirectSimulator2.xml, and testDirectSimulatorHierarchical.xml.

1.1.2 Verify correctness of model implementation

For small examples for which an analytical result can be calculated a unit test can be written to confirm the implementation behaves correctly for the expected result. For example, for a small tree ((A:1.0,B:1.0):1.0,(C:1.0,D:1.0):1.0) with birth rate 1 we can calculate the expected value of the Yule prior ($\log(P) = -6$), and write a unit test to make sure it matches:

```
package test;

import org.junit.Test;

import beast.evolution.speciation.YuleModel;
import beast.util.TreeParser;
import junit.framework.TestCase;

public class YuleLikelihoodTest extends TestCase {
```

```
    @Test
    public void testYuleLikelihood() {
        TreeParser tree = new TreeParser("((A:1.0,B:1.0):1.0,(C:1.0,D:1.0):1.0);");

        YuleModel likelihood = new YuleModel();
        likelihood.initByName("tree", tree, "birthDiffRate", "1.0");

        assertEquals(-6.0, likelihood.calculateLogP());
    }

}
```

In theory, the inferred distributions $p_I(\theta|M)$ should match the simulator distribution $p_S(\theta|M)$. However, drawing samples from $p_I(\theta|M)$ typically requires running an MCMC chain, which requires MCMC proposals R to randomly walk through state space. If we do this, we need to rely on R being correctly implemented. So, if we find that $p_I(\theta|M)$ and $p_S(\theta|M)$ do not match, it is not possible to tell whether problem is with an operator R or with the model implementation $I(M)$.

(Christiaan's technique to the rescue)

The Hastings ratio is $P(\theta)/P(\theta')$. Consequently, every proposal is accepted, whether $p_I(\theta|M)$ is correctly implemented or not.

In BEAST, if the `sample` method is implemented in a class derived from `Distribution`, you can use `beast.experimenter.DirectSimulatorOperator` in the Experimenter package to set up an MCMC analysis in XML. Here is an example that draws a birth rate from an exponential distribution with mean 1, and a Yule distribution to generate a tree. Note that the tree heigh statistic is logged, as well as an expression for a clock rate (being 0.5/tree-height) for evaluation purposes. The MCMC sample can be compared with the direct sample using the example file testDirectSimulator.xml.

```
<beast version="2.0" namespace="beast.core
:beast.evolution.alignment
:beast.evolution.tree
:beast.math.distributions
:beast.evolution.speciation
:beast.core.util
:beast.core.parameter">



    <run spec="MCMC" chainLength="1000000">
        <state id="state">
            <stateNode idref="tree"/>
            <stateNode idref="birthDiffRateParam"/>
```

```
        </state>

        <distribution spec="CompoundDistribution" id="fullModel">
            <distribution spec="YuleModel" id="yuleModel">
                <tree spec="Tree" id="tree">
                    <taxonset spec="TaxonSet">
                        <taxon spec="Taxon" id="t1"/>
                        <taxon spec="Taxon" id="t2"/>
                        <taxon spec="Taxon" id="t3"/>
                        <taxon spec="Taxon" id="t4"/>
                        <taxon spec="Taxon" id="t5"/>
                    </taxonset>
                </tree>
                <birthDiffRate spec="RealParameter" id="birthDiffRateParam" value="1.0"/>
            </distribution>

            <distribution spec="beast.math.distributions.Prior" id="birthDiffRatePrior">
                <distr spec="Exponential" id="xExpParamDist" mean="1"/>
                <x idref="birthDiffRateParam"/>
            </distribution>

        </distribution>

        <operator spec="beast.experimenter.DirectSimulatorOperator" weight="1" state="@state
            <simulator id="DirectSimulator" spec="beast.core.DirectSimulator" nSamples="1">
                <distribution idref="fullModel"/>
            </simulator>
        </operator>

        <logger id="tracelog" logEvery="1000" fileName="$(filebase).log">
            <log idref="birthDiffRateParam"/>
            <log id="clockRate" spec="beast.util.Script" expression="0.5/TreeHeight">
                <x id="TreeHeight" spec="beast.evolution.tree.TreeHeightLogger" tree="@tree'
            </log>
            <log idref="TreeHeight"/>
        </logger>

        <logger id="treelog" logEvery="1000" fileName="$(filebase).trees">
            <log idref="tree"/>
        </logger>

        <logger id="screenlog" logEvery="1000">
            <log idref="birthDiffRateParam"/>
        </logger>
    </run>
</beast>
```

Comparing two distributions can be done by

- eye balling the marginal likelihoods in Tracer and making sure they are close enough.
- testing whether parameters are covered 95% of the time in the 95% HPD interval of parameter distributions.
- using a statistical test, e.g. the Kolmogorov-Smirnov test, to verify the distributions $p_I(\theta|M)$ and $p_S(\theta|M)$ are the same.

`TraceKSStats` calculate Kolmogorov-Smirnof statistic for comparing trace logs. TraceKSStats has the following inputs:

- trace1 (LogFile): first trace file to compare (required)
- trace2 (LogFile): second trace file to compare (required)
- burnin (Integer): percentage of trace logs to used as burn-in (and will be ignored) (optional, default: 10)

Sample output:

Trace entry	p-value
posterior	1.0
likelihood	0.21107622404022763
prior	0.036794035181748064
treeLikelihood	0.04781117967724258
TreeHeight	0.036794035181748064
YuleModel	0.005399806065857771
birthRate	0.2815361702146215
kappa	0.62072545444263
freqParameter.1	0.0
freqParameter.2	8.930072172019798E-5
freqParameter.3	1.0883734952171764E-6
freqParameter.4	0.0

Though some values have very low p-values, meaning they differ significantly, it is recommended to verify this using Tracer to make sure that the test is not unduly influenced by outliers.

1.1.3 Verify correctness of operator implementations

Once simulator $S(M)$ and model implementation $I(M)$ are verified to be correct, next step is implementing efficient operators, running MCMCs to verify that parameters drawn from the prior are covered 95% of the time in the 95% HPD interval of parameter distributions.

The direct simulator operator (see `DirectSimulatorOperator` above) can be used as starting operator, and new operators added one by one to verify correctness.

The BEAST 2 Experimenter package can assist (see section "Using the Experimenter package" below).

1.2 Setting up a direct simulation in BEAST

Using the direct simulator can be done as follows

- Set up `DirectSimulator` at top level
- Add model to simulate from
- Add loggers to register output

1.2.1 Set up `DirectSimulator` at top level

Use the `nSamples` attribute to specify how many samples to draw.

```
<beast version="2.0" namespace="beast.core:beast.evolution.alignment:beast.evolution.tree:be

    <run spec="DirectSimulator" nSamples="100">

        <!-- model goes here -->

        <!-- loggers go here -->

    </run>
</beast>
```

1.2.2 Add model to simulate from

Here, the model consists of a birthDiffRate parameter, drawn from an exponential distribution. This rate is used in a Yule model to draw trees over a set of 5 taxa, called `t1`, `t2`,...,`t4`. This is placed inside the run element above.

```
<distribution spec="CompoundDistribution" id="fullModel">
    <distribution spec="YuleModel" id="yuleModel">
        <tree spec="Tree" id="tree">
            <taxonset spec="TaxonSet">
                <taxon spec="Taxon" id="t1"/>
                <taxon spec="Taxon" id="t2"/>
                <taxon spec="Taxon" id="t3"/>
                <taxon spec="Taxon" id="t4"/>
                <taxon spec="Taxon" id="t5"/>
            </taxonset>
        </tree>
        <birthDiffRate spec="RealParameter" id="birthDiffRateParam" value="1.0"/>
    </distribution>
```

```
    <distribution spec="beast.math.distributions.Prior" id="birthDiffRatePrior">
        <distr spec="Exponential" id="xExpParamDist" mean="1"/>
        <x idref="birthDiffRateParam"/>
    </distribution>
</distribution>
```

1.2.3 Add loggers to register output

Since we sample a parameter and a tree, we need a trace log and a tree log. Appart from state nodes, other statistics can be sampled as well. For example, here, we log the height of the tree using a `TreeHeightLogger`, and log a clock rate suitable for the tree using the expression `0.5/TreeHeight` using the `Script` class form the BEASTLabs package. The loggers should be placed inside the run element as well.

```
<logger logEvery="1" fileName="$(filebase).log">
    <log idref="birthDiffRateParam"/>
    <log id="TreeHeight" spec="beast.evolution.tree.TreeHeightLogger" tree="@tree"/>
    <log id="clockRate" spec="beast.util.Script" expression="0.5/TreeHeight">
        <x idref="TreeHeight"/>
    </log>
</logger>

<logger logEvery="1" fileName="$(filebase).trees">
    <log idref="tree"/>
</logger>
```

The complete XML file can be found as testDirectSimulatorByMCMC.xml in the Experimenter package.

1.3 Converting direct simulator XML to MCMC

Conversion requires the following steps:

- replace top level run element by MCMC
- add state element and references to state nodes
- add `DirectSimulatorOperator`
- add screen logger (optional)

2 Practical considerations

Validation only covers cases in as far as the prior covers it – most studies will not cover all possible cases, since the state space is just to large. Usually, informative

priors are required for validation to work, since broader priors (e.g. some of the default tree priors in BEAST) lead to identifiability issues, for example, due to saturation of mutations along branches of a tree.

3 Setting priors

3.1 Trees & clock model parameters

The mutation rate μ must have been such that the tree height h_T cannot exceed $1/\mu$ (in other words, $\mu h_T \leq 1$), otherwise there would be saturation, and sequences could not possibly have sufficient information to align. At the other end of the spectrum, where μh_T close to zero, very long sequences are required to ensure there are enough mutations in order to be able to reconstruct the tree distribution.

TODO: forumlate in terms of N_e instead of h_T?

- for reasonable computation times, trees should be about 0.5 substitutions high, OR
- sequences should be very long to reliably reconstruct smaller trees.

One way to enforce this is by

- a narrow prior on birth rates (for birth/death type tree priors), or
- putting an MRCA prior on the height of the tree, for coalescent models. Note that the latter hampers direct simulator implementations.

For clock models with mean clock rate != 1, simulate trees with clock rates times tree height approximately 0.5.

Published mutation rates can range from $O(1e-2)$ substitutions per site per year for viruses such as HIV (Cuevas *et al.* 2015), to $O(1e-11)$ for conserved regions of nuclear DNA (e.g PyrE2 locus in Haloferax volcanii (Lynch 2010)).

For releases, tree priors for clock and trees should be made less informative in order to cater for a wider range of tree heights and clock rates.

3.2 Gamma rate heterogeneity

To prevent saturation, adding categories with slow rates will go some way to allow covering a larger range of clock rates. Using gamma rate heterogeneity with shape values in the range 0.1 to 1 allows this, so adopt a gamma shape prior accordingly.

3.3 Proportion invariable sites

Since each site evolves with non-zero rate, use of proportion invariable sites is modeling the process badly, and therefore not recommended.

3.4 Frequencies

Priors ideally should be set in realistic ranges, e.g. frequency priors not uniform(0,1) but Dirichlet(4,4,4,4) is better.

3.5 Substitution model parameters

Default priors seem OK for most substitution models.

3.6 Sequence simulator

`SequenceSimulator` can help generate individual alignments.

To generate N XML files, use `CoverageTestXMLGenerator` in Experimenter package

The sequence length should be long enough that trees can be reasonably reliably recovered – if the difference between longest and shorted tree is 2 orders of magnitude, nucleotide sequences of 10 thousand sites. When μ*treeh-height approximate 0.5, sequences of length 1000 are sufficient.

3.7 Log file names

Make sure log files names do not overlap. Use `logFileName="out$(N).log` and start BEAST with `for i in {0..99} do /path/to/beast/bin/beast -D N=$i beast$i.xml; done`

4 Trouble shooting

4.1 Coverage gone wrong

One reason coverage can be lower is if the ESSs are too small, which can be easily checked by looking at the minimum ESS for the log entry. If these values are much below 200 the chain length should be increased to be sure any low coverage is not due to insufficient convergence of the MCMC chain.

4.2 Low coverage

The occasional 91 is acceptable (the 95% HPD = 90 to 98 probability the implementation is correct) but coverage below 90 almost surely indicate an issue with the model or operator implementation. Also, coverage of 99 or 100 should be looked at with suspicion – it may indicate overly wide uncertainty intervals.

If correct, distributed binomial with p=0.95, N=100:

k	p(x=k)	p(x<=k)	p(x>=k)
90	0.0167	0.0282	0.9718
91	0.0349	0.0631	0.9369
92	0.0649	0.1280	0.8720
93	0.1060	0.2340	0.7660
94	0.1500	0.3840	0.6160
95	0.1800	0.5640	0.4360
96	0.1781	0.7422	0.2578
97	0.1396	0.8817	0.1183
98	0.0812	0.9629	0.0371
99	0.0312	0.9941	0.0059
100	0.0059	1.0000	0.0000

source https://www.di-mgt.com.au/binomial-calculator.html for different values of p and N.

4.3 Common causes of low coverage

- trees cannot be reconstructed reliably (height should not be too small or large).
- Hastings ratio in operators incorrectly implemented
- bug in model likelihood

5 Releasing

5.1 Setting priors in BEAUti template

- Priors should be made as uninformative as possible – people will use defaults!
- Make sure to consider a number of scenarios, e.g. for clock models, scenarios from setting up rates page on beast2.org.

5.2 Communicating results

When publishing well calibrated studies, all XML files, log files and scripts for manipulating them should be made available, so the study can be replicated exactly as is. A practical way to do this is through a github repository.

Version numbers of BEAST and packages used should be noted.

6 Using the Experimenter package

Experimenter is a BEAST 2 package that assists in simulation studies to verify correctness of the implementation. The goal of this particular simulation studies is to make sure that the model or operator implementation is correct by running N analysis on simulated data (using SequenceSimulator) on a tree and site model parameters sampled from a prior.

To run a simulation study:

- set up XML for desired model and sample from prior
- generate (MCMC) analysis for each of the samples (say 100)
- run the analyses
- use loganalyser to summarise trace files
- run CoverageCalculator to summarise coverage of parameters

Make sure to have the Experimenter package installed (details at the end).

6.1 1. Set up XML for desired model and sample from prior

First, you set up a BEAST analysis in an XML file in the configuration that you want to test. Set the `sampleFromPrior="true"` flag on the element with MCMC in it, and sample from the prior. Make sure that the number of samples in the trace log and tree log is the same and that they are sampled at a frequency such that there will be N useful samples (say N=100).

6.2 2. Generate (MCMC) analysis for each of the samples

You can use CoverageTestXMLGenerator to generate BEAST XML files from a template XML file. The XML file used to sample from the prior can be used for this (when setting the sampleFromPrior flag to false). You can run CoverageTestXMLGenerator using the BEAST applauncher utility (or via the `File/Launch Apps` meny in BEAUti).

CoverageTestXMLGenerator generates XML for performing coverage test (using CoverageCalculator) and has the following arguments:

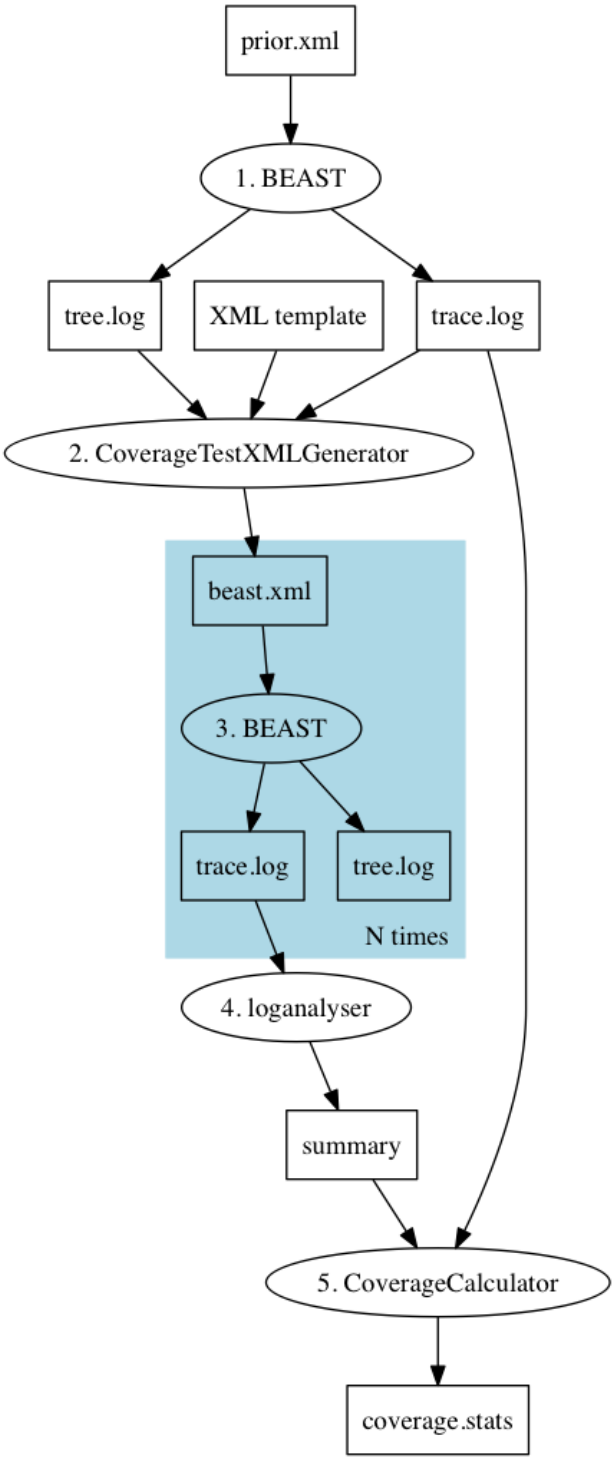

Figure 1: Summary of files involved in testing an operator. Rectangles represent files, ovals represent programs.

-workingDir working directory where input files live and output directory is created -outDir output directory where generated XML goes (as sub dir of working dir) (default: mcmc) -logFile trace log file containing model paramter values to use for generating sequence data -treeFile tree log file containing trees to generate sequence data on -xmlFile XML template file containing analysis to be merged with generated sequence data -skip numer of log file lines to skip (default: 1) -burnin percentage of trees to used as burn-in (and will be ignored) (default: 1) -useGamma [true|false] use gamma rate heterogeneity (default: true) -help show arguments

`NB: make sure to set sampleFromPrior="false" in the XML.`

`NB: to ensure unique file name, add a parameter to logFileName, e.g. logFileName="out$(N).log"`

With this setting, when you run BEAST with `-D N=1` the log file will `be out1.log`.

6.3 3. Run the analyses

Use your favourite method to run the N analyses, for example with a shell script

```
for i in {0..99} do /path/to/beast/bin/beast -D N=$i beast$i.xml; done
```

where `/path/to/beast` the path to where BEAST is installed.

6.4 4. Use loganalyser to summarise trace files

Use the loganalyser utility that comes with BEAST in the bin directory. It is important to use the `-oneline` argument so that each log line gets summarised on a single line, which is what `CoverageCalculator` expects. Also, it is important that the log lines are in the same order as the log lines in the sample from the prior, so put the results for single digits before those of double digits, e.g. like so:

```
/path/to/beast/bin/loganalyser -oneline out?.log out??.log > results
```

where `out` the base name of your output log file.

6.5 5. Run `CoverageCalculator` to summarise coverage of parameters

You can run CoverageCalculator using the BEAST applauncher utility (or via the `File/Launch Apps` meny in BEAUti).

CoverageCalculator calculates how many times entries in log file are covered in an estimated 95% HPD interval and has the following arguments:

- log log file containing actual values
- skip numer of log file lines to skip (default: 1)
- logAnalyser file produced by loganalyser tool using the -oneline option, containing estimated values
- out output directory for tsv files with truth and estimated mean and 95% HPDs, and directory is also used to generate svg bargraphs and html report. Not produced if not specified.
- help show arguments

It produces a report like so:

```
coverage Mean ESS Min ESS posterior                                    0
2188.41  1363.02 likelihood                                            0
4333.99  3042.15 prior                                                 33
1613.20  891.92 treeLikelihood.dna                                     0
4333.99  3042.15 TreeHeight                                            95
3076.44  2233.29 popSize                                               94
577.20  331.78 CoalescentConstant                                     91
1620.76  787.30 logP(mrca(root))                                       97
4320.70  3328.88 mrca.age(root)                                        95
3076.44  2233.29 clockRate                                            0
3046.64  2174.60 freqParameter.1                                       98
4332.76  3388.90 freqParameter.2                                       97
4337.93  3334.29 freqParameter.3                                       96
4378.30  3462.73 freqParameter.4                                       92
4348.83  3316.36
```

Coverage should be around 95%. One reason coverage can be lower is if the ESSs are too small, which can be easily checked by looking at the `Mean ESS` and `Min ESS` columns. If these values are much below 200 the chain length should be increased to be sure any low coverage is not due to insufficient convergence of the MCMC chain. The occasional 90 or 91 is acceptable but coverage below 90 almost surely indicate an issue with the model or operator implementation.

The values for posterior, prior and treelikelihood can be ignored: it compares results from sampling from the prior with that of sampling from the posterior so they can be expected to be different.

If an output file is specified, `CoverageCalcaulator` also generates an HTML file with bar graphs (in svg) showing how well each item in the log file covers the true value, as well as tab separated (tsv) files containing the data, so you can import them in for example R to produce customised graphs. Below some examples with good coverage, and strong, medium and weak ability to learn the parameter, followed by over estimated and under estimated parameters.

Graphs show true value on x-axis and estimates on y-axis. Black line shows where x equals y axis, and where ideally most of the probability mass is concentrated. Black dots are means of estimates. Bars indicate 95% HPDs where blue bars

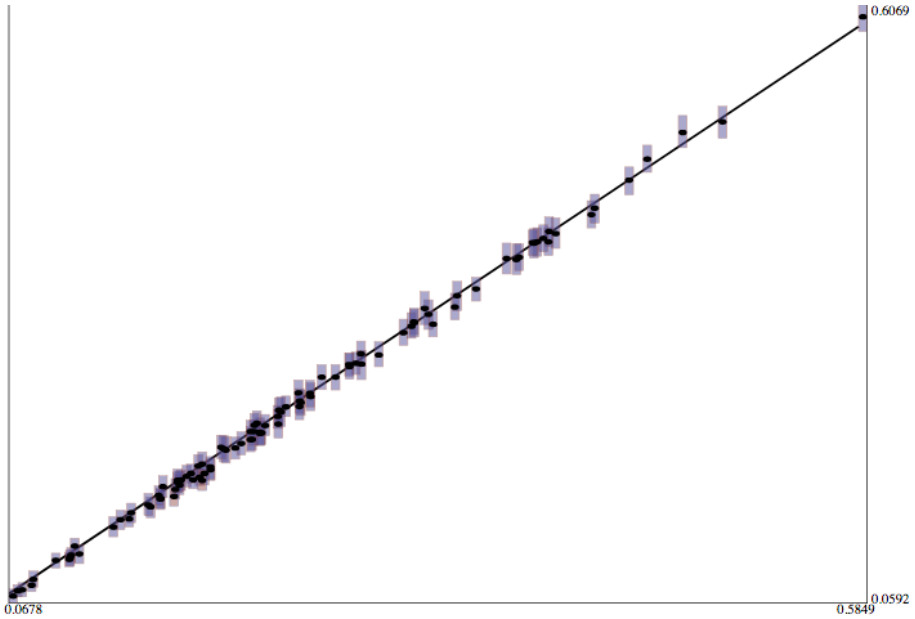

Figure 2: Strong coverage: true parameter can be inferred accurately.

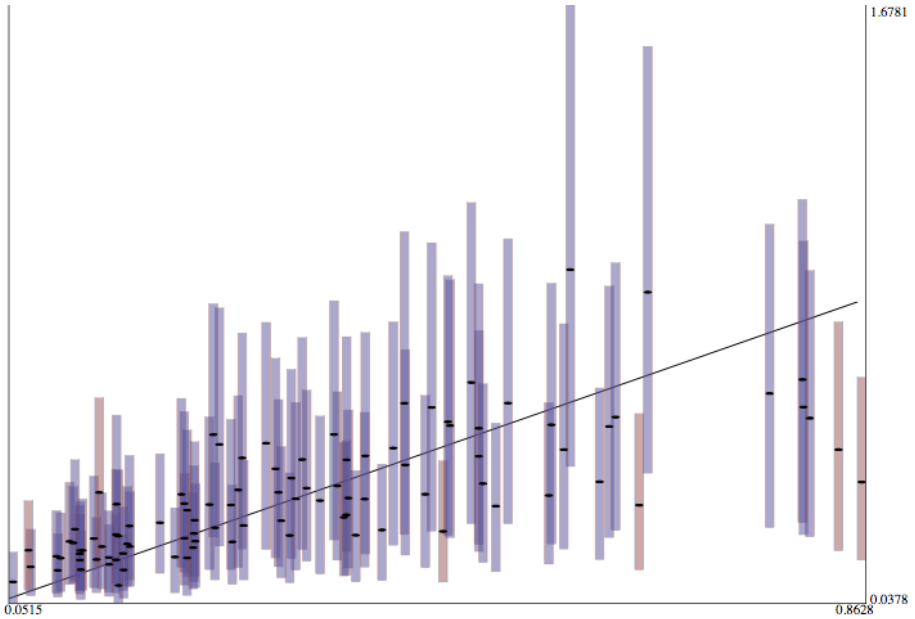

Figure 3: Medium coverage: true parameter can be inferred, but with high uncertainty.

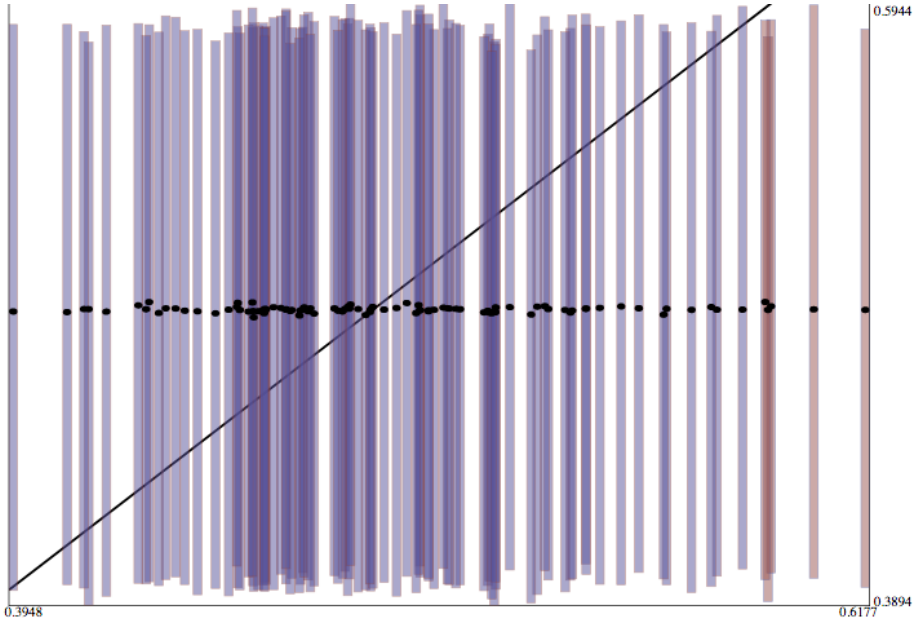

Figure 4: Weak coverage: true parameter cannot be inferred, even though 95% HPD covers the true value sufficiently often.

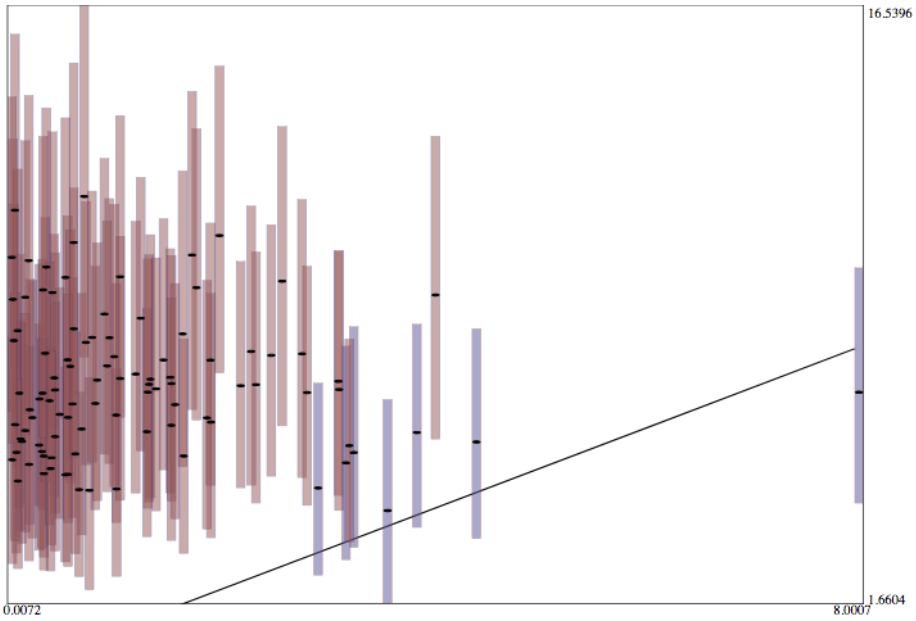

Figure 5: True parameter is over estimated

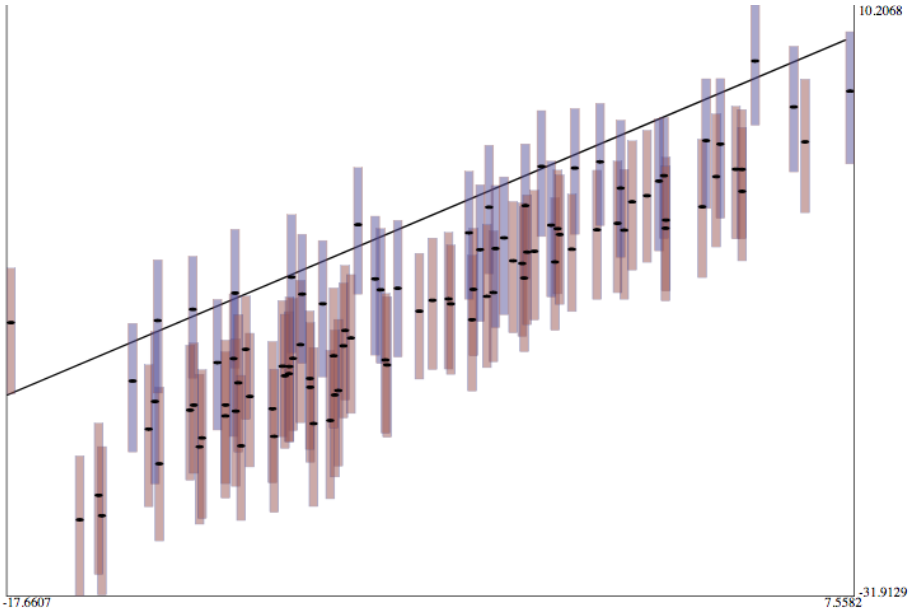

Figure 6: True parameter is under estimated

cover the true value and red ones do not. Ideally 95 out of 100 bars should be blue.

6.6 Simulation Based Calibration

Simulation Based Calibration (SBC) (Talts *et al.* 2018) is a way to validate how well true values used to generate data rank inside the inferred distributions. Ranks are binned, and the resulting bins should be uniformly distributed if all is well. Deviation from uniform distributions indicate

- if shaped like a U the posterior is too narrow.
- if shape like inverted U, the posterior is too wide.
- if shaped sloping upwards, the posterior is biased towards lower estimates.
- if shaped sloping downwards, the posterior is biased towards higher estimates.

To run a simulated based calibration study (steps 1-3 as for a coverage study):

- set up XML for desired model and sample from prior
- generate (MCMC) analysis for each of the samples (say 100)
- run the analyses
- use `LogAnalyser` to find minimum ESS
- run `LogCombiner` to sub sample log files and accumulate logs
- run `SBCAnalyser` to summarise coverage of parameters

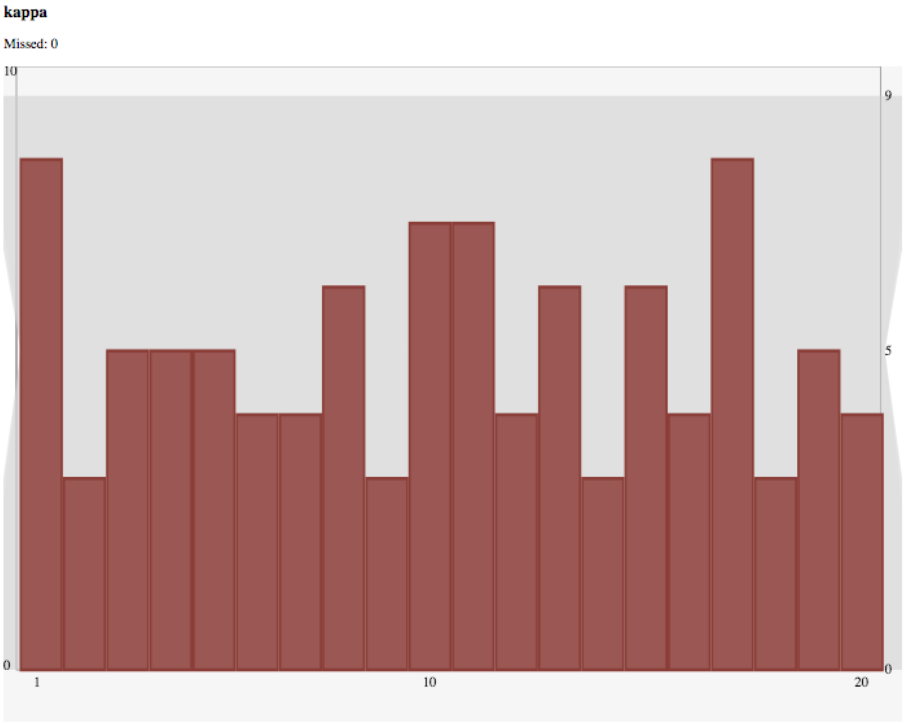

Figure 7: Simulation based calibration output for kappa parameter with 20 intervals. Light grey band indicates 99% coverage interval and darker grey the 95% coverage interval.

A correct implementation is uniformly distributed, like so:

For steps 1-3, see coverage study.

6.7 4. Find minimum ESS

Use `LogAnalyser` to find minimum ESS – or run coverage study and minimum ESS will be printed as part of the analysis.

6.8 5. Combine logs

First, we need to determine how much to resample log files. Since samples must be independent for the method to work, we can resample with frequency equal to the chain length divided by minimum ESS.

Run `LogCombiner` to sub sample log files and accumulate logs. To run from command line, use

```
/path/to/beast/bin/logcombiner -resample <resample> -log <name>-?.log <name>-??.log <name>-?
```

where `<resample>` is the resample frequency (= chain length/minium ESS), and `<name>-` the name of the log file. Note that if you numbered the log files 0,...,9,10,...,99,100,...,999 using `<name>-*.log` will put entries in an alphabetic order, which is probably *not* what you want.

6.9 6. Run `SBCAnalyser` to summarise coverage of parameters

`SBCAnalyser` can be run with the BEAST app launcher, and outputs a report and (if an output directory is specified). It has the following arguments:

- SBCAnalyser has the following inputs:
- log (File): log file containing actual values (required)
- skip (Integer): numer of log file lines to skip (optional, default: 1)
- logAnalyser (File): file produced by loganalyser tool using the -oneline option, containing estimated values (required)
- bins (Integer): number of bins to represent prior distribution. If not specified (or not positive) use number of samples from posterior + 1 (L+1 in the paper) (optional, default: -1)
- outputDir (OutFile): output directory for SVG bar charts (optional, default: [[none]])
- useRankedBins (Boolean): if true use ranking wrt prior to find bins.if false, use empirical bins based on prior. (optional, default: true)

Note that it compares entries from the prior to posterior, so items like likelihood, posterior, treeLikelihood and clockRate seem very wrong, but that can be ignored,

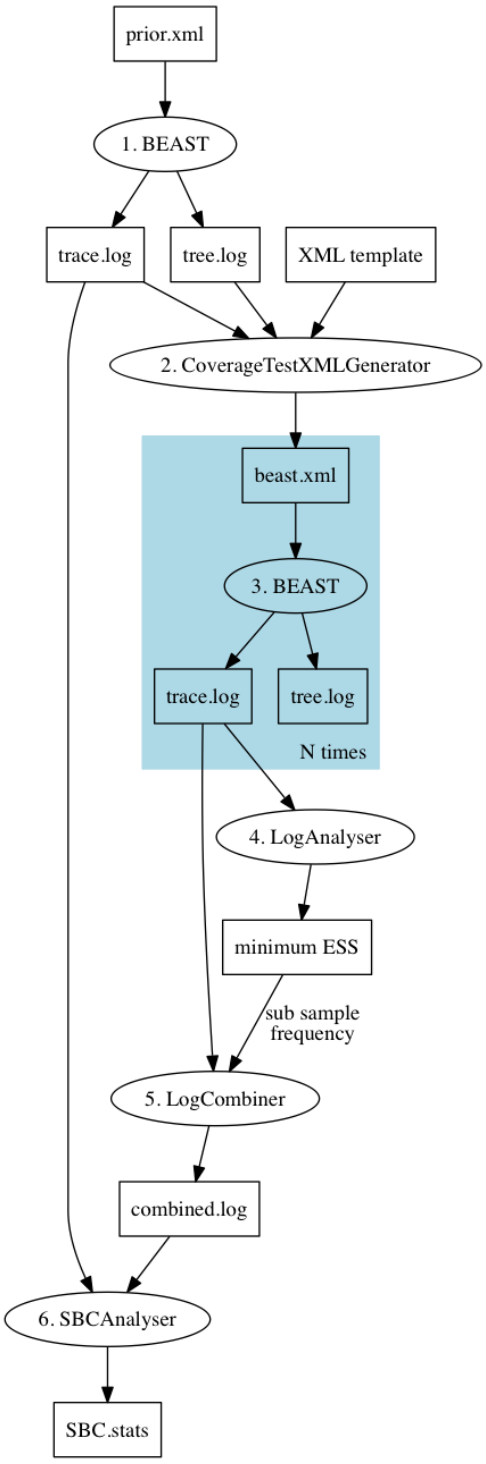

Figure 8: Summary of files involved in running a simulation based calibration study. Rectangles represent files, ovals represent programs.

since these were not part of the prior or (for clockRate) we know beforehand the prior differs substantially from the posterior.

```
99%lo << mean << 99%up = -1 << 5 << 10
```

	missed		bin0		bin1		bin2		bin3	
posterior	1	0	0	0	0	0	0	0	0	0
likelihood	1	0	0	0	0	0	0	0	0	0
prior	0	8	5	6	6	4	3	7	5	1
treeLikelihood.dna	1	0	0	0	0	0	0	0	0	0
TreeHeight	1	3	5	3	3	2	4	5	3	4
kappa	0	8	3	5	5	5	4	4	6	3
gammaShape	0	6	6	2	4	3	2	2	6	6
popSize	1	1	3	1	3	2	6	1	3	1
CoalescentConstant	0	7	5	0	7	9	5	5	4	5
parameter.hyperInverseGamma-beta-PopSizePrior	0	6	3	2	6	3	3	3	7	3
HyperPrior.hyperInverseGamma-beta-PopSizePrior	1	11	2	5	4	5	3	5	10	1
monophyletic(root)	1	0	0	0	0	0	0	0	0	0
logP(mrca(root))	0	5	5	1	4	2	4	6	5	4
mrca.age(root)	1	3	5	3	3	2	4	5	3	4
clockRate	1	0	0	0	0	0	0	0	0	0
freqParameter.1	1	4	4	7	3	6	7	4	6	1
freqParameter.2	0	6	9	3	5	4	7	4	4	1
freqParameter.3	0	6	4	6	8	8	5	4	5	4
freqParameter.4	0	9	1	2	5	4	3	4	6	3

```
Done!
```

6.10 Installing Experimenter package

Currently, you need to build from source (which depends on BEAST 2, BEASTlabs and MASTER code) and install by hand (see "install by hand" section in managing packages.

Quick guide

- clone BEAST 2, BEASTlabs, MASTER, and Experimenter all in same directory.
- build BEAST 2 (using `ant Linux` in the beast2 folder), then BEASTLabs (using `ant addon` in the BEASTLabs folder), and MASTER (using `ant` in the MASTER folder) then Experimenter (again, using `ant addon` in the Experimenter folder) packages.
- install BEASTlabs (using the package manager, or via BEAUti's `File/Manage pacakges` menu).
- install Experimenter package by creating `Experimenter` folder in your BEAST package folder, and unzip the file `Experimenter/build/dist/Experimenter.addon.v0.0.1.zip` (assuming version 0.0.1).

References

Bouckaert R, Vaughan TG, Barido-Sottani J *et al.* BEAST 2.5: An advanced software platform for bayesian evolutionary analysis. *PLoS computational biology* 2019;**15**:e1006650.

Bouckaert RR, Heled J, Kühnert D *et al.* BEAST 2: A software platform for Bayesian evolutionary analysis. *PLoS Comput Biol* 2014;**10**:e1003537.

Bryant D, Bouckaert R, Felsenstein J *et al.* Inferring species trees directly from biallelic genetic markers: Bypassing gene trees in a full coalescent analysis. *Molecular biology and evolution* 2012;**29**:1917–32.

Cuevas JM, Geller R, Garijo R *et al.* Extremely high mutation rate of hiv-1 in vivo. *PLoS biology* 2015;**13**:e1002251.

Drummond AJ, Bouckaert RR. *Bayesian Evolutionary Analysis with BEAST*. Cambridge: Cambridge University Press, 2015.

Lynch M. Evolution of the mutation rate. *TRENDS in Genetics* 2010;**26**:345–52.

Talts S, Betancourt M, Simpson D *et al.* Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:180406788* 2018.

Vaughan TG, Drummond AJ. A stochastic simulator of birth–death master equations with application to phylodynamics. *Molecular biology and evolution* 2013;**30**:1480–93.