BaSTA

an R package for Bayesian estimation of age-specific survival from incomplete mark-recapture/recovery data with covariates

Fernando Colchero, Owen R. Jones and Maren Rebke
Max Planck Institute for Demographic Research

Contents

1 Introduction

Here we present BaSTA (Bayesian Survival Trajectory Analysis), a free open-source R package (R Development Core Team 2011) that implements the hierarchical Bayesian model described by Colchero & Clark (2011). This package facilitates drawing inference on age-specific mortality from capture-recapture/recovery (CRR) data when a large proportion (or all) of the records have missing information on times of birth and death. In addition, BaSTA allows users to evaluate the effect of continuous and categorical covariates on mortality.

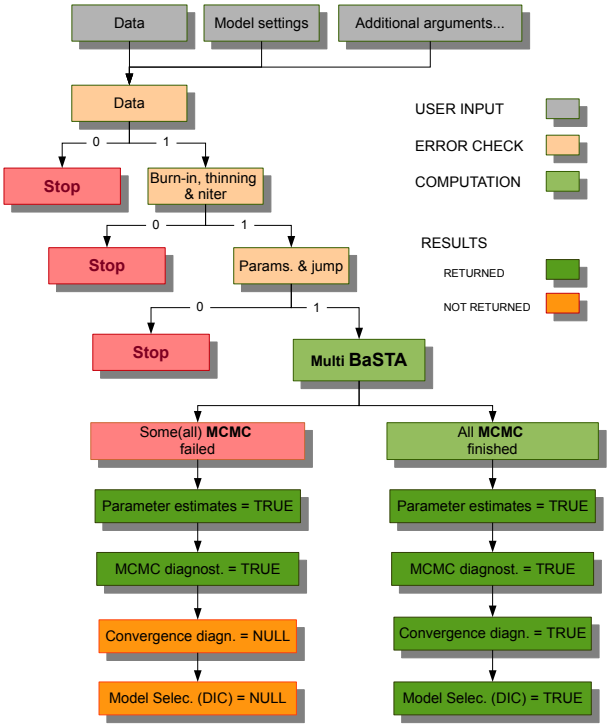

Figure 1: Work flow of BaSTA's main function `basta()` after data input and argument definition from the user. During the initial "Error check" sequence, 1 implies that no errors were detected and 0 means otherwise. In the later case, the function is stopped and an error message is printed explaining the error and suggesting solutions.

BaSTA consists of a set of routines initialised by the user through data input and the definition of basic model settings (Fig. **??**). The package then verifies that the data has the right format, and that the user-defined model settings are consistent (i.e. initial error checks). If no errors are found, the model runs one or multiple Markov Chain Monte Carlo (MCMC) algorithms (for a full description of the algorithm see Colchero & Clark 2011). After the MCMC runs are finished, the package calculates a range of diagnostics that include measures of serial autocorrelation on parameter traces, parameter update rates, convergence and preliminary model selection.

The package's main function, called `basta()`, defines its own S3 method (Chambers & Hastie 1992) and outputs an object of class `basta` which can then be explored with the generic `plot()`, `summary()` and `print()` functions. The package also includes two data formatting functions, `CensusToCaptHist()` and `MakeCovMat()`, and a data checking function `DataCheck()`.

2 Data formatting

BaSTA's input data format is compatible with other programs that deal with CRR data sets such as MARK (White & Burnham 1999). The data needs to be configured as a table in data frame format where each row corresponds to one individual. The first column corresponds to the individual IDs while the second and third columns give the years of birth and death, respectively. Next, T columns (T = study span), one per study year, are filled with the individual recapture histories. Thus, every year an individual is detected the corresponding column is filled with 1 and 0 otherwise.

If covariates are to be included, additional columns can be added, with one column per covariate. For instance, table 1 shows a data frame for four individuals and a study span of $T = 4$ years with one covariate, i.e. location. In this example, individuals 1 and 2 have known birth year and individuals 1 and 3 have know death year, individual 1 was detected in the first, third and fourth year of the study, individual 2 from the second to the forth year and so forth. Individuals 1 and 2 belong to location 1 and individuals 3 and 4 to location 2.

Table 1: Data format required by BaSTA.

ID	Birth	Death	year 1	year 2	year 3	year 4	location
1	b_1	d_1	1	0	1	1	loc 1
2	b_2	0	0	1	1	1	loc 1
3	0	d_3	1	1	1	1	loc 2
4	0	0	0	1	1	0	loc 2

2.1 Construct capture history matrix: `CensusToCaptHist()`

This capture history matrix can be constructed using BaSTA's built-in data formatting functions. For instance, with function `CensusToCaptHist()`, it is possible to convert a conventional individual survey table (i.e. one record per time an individual is observed) to the recapture matrix described above. Below is an example with a simulated capture history of five individuals between 1990 and 2000:

```
> id.vec <- sort(c(rep(1, 3), rep(2, 2), rep(3, 4), rep(4, 3),
+     rep(5, 3)))
> d.vec <- rep(0, length(id.vec))
> for (i in unique(id.vec)) {
+     svec <- which(id.vec == i)
+     d.vec[svec] <- sort(sample(1990:2000, length(svec)))
+ }
> Y <- CensusToCaptHist(ID = id.vec, d = d.vec, dformat = "yyyy")
```

The `ID` argument in the `CensusToCaptHist()` function, takes a vector with the individual IDs, along with the `d` argument, which is a vector of dates on which each individual was detected, and argument `dformat` which is (optionally) used to specify the date format used in `d`.

The resulting capture history matrix looks like this:

```
   yr
ID  1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
  1    0    1    1    0    0    0    0    0    0    1    0
  2    0    0    0    0    0    0    0    1    1    0    0
  3    0    0    1    0    0    0    0    1    1    0    1
  4    1    0    1    1    0    0    0    0    0    0    0
  5    0    0    0    1    1    0    1    0    0    0    0
```

2.2 Construct covariates matrix: `MakeCovMat()`

Covariates can be set up in the apropriate format with the `MakeCovMat()` function. Below is an example with simulated data for five individuals:

```
> sex <- sample(c("f", "m"), 5, replace = TRUE)
> weight <- rnorm(5, mean = 10, sd = 1)
> raw.mat <- data.frame(sex, weight)
```

This covariate data frame looks like this:

```
  sex    weight
1   m  9.229235
2   m  8.928843
3   m 10.411781
4   m  9.373837
5   f  9.441571
```

The data frame `raw.mat` contains one column for sex, a categorical covariate, and one for weight, a continuous covariate. This data frame can be then rearranged into a suitable format for BaSTA with the `MakeCovMat()` function. The data can then be incorporated into the main data frame used as an input for BaSTA:

```
> cov.mat <- MakeCovMat(x = c("sex", "weight"), data = raw.mat)
```

which produces the following matrix:

```
  sexf sexm    weight
1    0    1  9.229235
2    0    1  8.928843
3    0    1 10.411781
4    0    1  9.373837
5    1    0  9.441571
```

Argument `x` can be used to specify which covariates should be included in the covariate matrix either with a character string vector (see example above), or with a numerical vector that indicates the column numbers in the data frame `data` that should be used for inference. If all of the columns are to be included, the `x` argument can be ignored and only the `data` argument needs to be specified. Alternatively, `x` can be of class `formula`, as we show in the following example:

```
> cov.mat <- MakeCovMat(x = ~sex + weight + sex:weight, data = raw.mat)
```

which produces the following matrix:

```
  sexf sexm    weight sexm:weight
1    0    1  9.229235    9.229235
2    0    1  8.928843    8.928843
3    0    1 10.411781   10.411781
4    0    1  9.373837    9.373837
5    1    0  9.441571    0.000000
```

In this case, we are also including an interaction between sex and weight. For further details on how to specify a formula in R, type `help(formula)` in the R console.

2.3 Verify data consistency: `DataCheck()`

After the final data frame is constructed, it can be verified with the `DataCheck()` function. This function performs a range of diagnostic checks on the data frame (Table 2). Below is an example with the simulated dataset we provide with the package:

```
> data("sim1.dat", package = "BaSTA")
> new.dat <- DataCheck(sim1.dat, studyStart = 51, studyEnd = 70,
+     autofix = rep(1, 7), silent = FALSE)
```

If the `silent` argument is set as `FALSE`, then the function prints out a range of descriptive statistics about the dataset, exemplified as follows:

```
No problems were detected with the data.


*DataSummary*
- Number of individuals         =   2,600
- Number with known birth year  =    979
- Number with known death year  =    385
- Number with known birth
 AND death years               =    385

- Total number of detections
 in recapture matrix           =  10,285

- Earliest detection time       =      51
- Latest detection time         =      70
- Earliest recorded birth year  =      51
- Latest recorded birth year    =      70
- Earliest recorded death year  =      52
- Latest recorded death year    =      91
```

`DataCheck()` searches the dataset for seven different types of error (Table 2), which can be fixed using argument `autofix`. Although this can save a lot of time and effort to the user, we strongly advise users to verify the reported errors and make an informed decision on how to fixed them.

As an example, we show below how `DataCheck()` reports errors for a dataset that includes death years that apparently occur before the year of birth (i.e. a **type 3** error) for a few individuals:

Table 2: Description of error types in datasets as defined in the `DataCheck()` function and the actions that are taken based on the values provided in argument `autofix`.

Error type	Description	autofix code
type 1	Deaths occurring before the study starts	0 = do nothing; 1 = remove from dataframe
type 2	No birth/death AND no recaptures	0 = do nothing; 1 = remove from dataframe
type 3	Births recorded after death	0 = do nothing; 1 = replace death records with 0; 2 = replace birth records with 0; 3 = replace both birth and death records with 0
type 4	Recaptures after death	0 = do nothing; 1 = remove spurious post-death observations
type 5	Recaptures before birth	0 = do nothing; 1 = remove observations that pre-date year of birth
type 6	Year of birth is not a zero in the recapture matrix	0 = do nothing; 1 = replace birth year element of observation matrix with 0
type 7	Year of death is not a zero in the recapture matrix	0 = do nothing; 1 = replace death year element of observation matrix with 0

```
> new.dat <- DataCheck(dat.error, studyStart = 51, studyEnd = 70,
+     autofix = rep(1, 7), silent = TRUE)

The following rows have birth dates that are later than their death dates:
[1] 1169 1641 2111
The death records have been replaced with 0.
```

3 Setting up the analysis: function `basta()`

After the data has been formatted and verified for consistency, the analysis is performed with the `basta()` function. In this section we explain the arguments used in this function. This function can be run, in it's simplest form, by specifying only the dataset (with the `object` argument), and the start and end times of the study with the `studyStart` and `studyEnd` arguments. Thus, a simple analysis for a study starting in 1990 and finishing in 2000 and with a hypothetical data frame called '`mydata`' can be performed by entering the following command:

```
> out <- basta(object = mydata, studyStart = 1990, studyEnd = 2000)
```

All of the other arguments in the `basta()` function have default values that allow users to run the model without specifying any additional information. The default values can be viewed in the `basta()` help file with the comment `?basta`. In order to take advantage of the full functionality of BaSTA we recommend that users explore different models and shapes, as well as a variety of covariate structures. Below we outline how to set up an analysis with BaSTA in order to test a range of models and covariate data structures (further details on the models and diagnostics can be found in Colchero *et al.* 2011).

3.1 Choosing mortality models: arguments `model` and `shape`

The `model` argument can be used to choose between four basic mortality rate functions: a) '`EX`'; exponential (Cox & Oakes 1984); b) '`GO`' (default); Gompertz (Gompertz 1825, Pletcher 1999); c) '`WE`'; Weibull (Pinder III *et al.* 1978); and d) '`LO`'; logistic (Pletcher 1999) (Table 4). Each one of these functions can describe different trends in age-specific mortality, giving BaSTA considerable flexibility when estimating these vital rates (Fig. **??**).

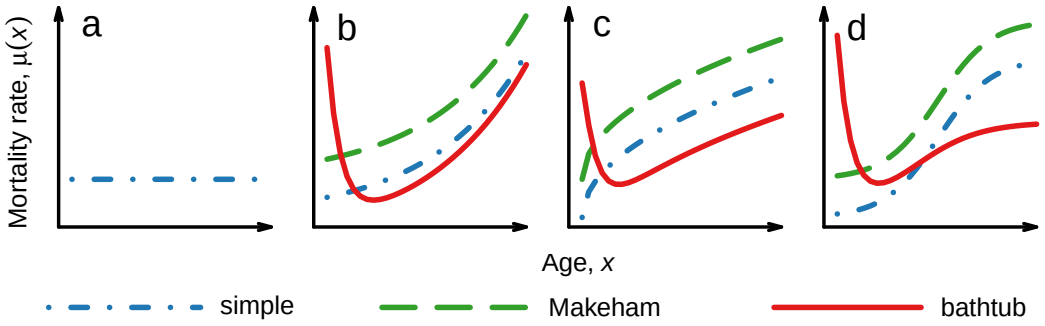

Figure 2: Mortality rates, $\mu(x|\theta)$, resulting from the four basic models included in BaSTA: a) exponential; b) Gompertz; c) Weibull; and d) logistic. The three different lines in each plot (except in a) show examples of the shapes that can be tested with BaSTA, namely: 'simple'; 'Makeham'; and 'bathtub'.

In addition, BaSTA allows users to extend these basic functions in order to examine more complex shapes. Specifically, three general forms can be defined with the `shape` argument: i) '`simple`' (the default shape), which uses only the basic functions defined in Table 4; ii) '`Makeham`' (Pletcher 1999), which adds a constant to the mortality rate; and iii) '`bathtub`' (e.g. Siler 1979), which consists of adding a declining Gompertz function and a constant to the basic mortality rate. The resulting shapes can be seen in Fig. **??**. Clearly, the number of parameters used in each of these combinations varies. In table 3 we show the number of parameters for the different types of mortality models and shape combinations.

Table 3: Number of parameters for all combinations of mortality models and shapes that can be tested in BaSTA.

Model	simple	Makeham	bathtub
Exponential	1	–	–
Gompertz	2	3	5
Weibull	2	3	5
Logistic	3	4	6

For example, to run the analysis using a logistic mortality rate with a bathtub shape, the specifications for the function should be:

```
> out <- basta(object = mydata, studyStart = 1990, studyEnd = 2000,
+     model = "LO", shape = "bathtub")
```

If the model or the shape are misspecified, the analysis will be stopped and an error message is printed that clarifies which argument values should be used.

3.2 Defining covariate structure: the `covarsStruct` argument

BaSTA also allows to evaluate the effect of covariates on age patterns of mortality. This is achieved with the `covarsStruct` argument, which defines three optional structures: i) '`fused`' (default), in which covariates are separated into continuous and categorical, and where the former are included into a proportional hazards framework (Klein & Moeschberger 2003), and the latter are included as linear functions of the mortality parameters. This is analogous to their treatment in generalized linear models (GLMs); ii) '`prop.haz`' where all covariates are included as proportional hazards; and iii) '`all.in.mort`' where all covariates are evaluated as linear functions of the mortality parameters. Currently, the latter structure can only be implemented with a Gompertz ('`GO`') model with a '`simple`' shape.

3.3 MCMC general settings: the `niter`, `burnin` and `thinning` arguments

The number of MCMC steps can be specified with the `niter` argument, while the burn-in sequence and the thinning interval are controlled with arguments `burnin` and `thinning`, respectively. The burn-in corresponds to the initial sequence before parameters reach convergence, which is commonly discarded, leaving the remaining steps to calculate a range of diagnostics and other statistics (Clark 2007). The thinning interval is set in order to reduce serial autocorrelation between consecutive parameter estimates. Based on the results from Colchero & Clark (2011), the default values are `niter` $= 50,000$ steps, `burnin` $= 5,001$ and `thinning` $= 50$. Still, we recommend that these values should be tested before the final simulations are implemented.

3.4 Initial parameters, jumps and priors

Although BaSTA has built-in default values for initial parameters, jump standard deviations, and priors, these can be modified with the arguments `thetaStart` and `gammaStart` for mortality and proportional hazards initial parameters, and the corresponding `thetaJumps`, `thetaPriors`, `gammaJumps` and `gammaPriors` arguments for jumps and priors. It is important to note that the length of the vector or the dimensions of the matrices specified should correspond to the number of parameters for each combination of `model`, `shape`, and `covarsStruct`. For instance, if a logistic ('`LO`') model with '`simple`' shape (i.e. 3 parameters, Table 3) and a '`mixed`' covariate structure is chosen, and two categorical and two continuous covariates are included in the dataset, `thetaStart`, `thetaJumps` and `thetaPriors` should be vectors of length 3 (the same set of parameters for both categorical covariates), or of length 6 (one set of parameters per covariate), or matrices of dimension 2×3. Also, `gammaStart`, `gammaJumps` and `gammaPriors` should all be vectors of length 2 for this example. For example, if we wish to specify the jumps for the mortality parameters in this example, we could type:

```
> out <- basta(object = mydata, studyStart = 1990, studyEnd = 2000,
+     model = "LO", shape = "simple", thetaJumps = c(0.1, 0.1,
+         0.1))
```

or, alternatively we could create a matrix of jumps of the form:

```
> new.jumps <- matrix(c(rep(0.1, 3), rep(0.2, 3)), nrow = 2, ncol = 3,
+     byrow = TRUE, dimnames = list(c("cov1", "cov2"), paste("b",
+         0:2, sep = "")))
```

where each column corresponds to a mortality parameter, and each row to a covariate, of the form:

```
     b0  b1  b2
cov1 0.1 0.1 0.1
cov2 0.2 0.2 0.2
```

which then we could use for the `thetaJumps` argument:

```
> out <- basta(object = mydata, studyStart = 1990, studyEnd = 2000,
+     model = "LO", shape = "simple", thetaJumps = new.jumps)
```

3.5 Multiple runs: arguments `nsim`, `parallel` and `ncpus`

To ensure that parameter estimates derived from MCMC routines converge appropriately, it is necessary to run several simulations from over-dispersed initial parameter values (Gelman *et al.* 2004). By doing this, it is possible to confirm whether the parameter chains (i.e. traces) all converge to the same final values, irrespective of the initial parameters. BaSTA allows users to run multiple simulations by specifying the number of runs desired with `nsim` argument. Moreover, to reduce the amount of computing time, BaSTA facilitates the performance of these multiple runs in parallel using the `snowfall` package (Knaus 2010). This is achieved by setting the logical argument `parallel` as '`TRUE`'. In addition, the number of cores used can be selected with argument `ncpus`. If the package `snowfall` is not installed, or the argument `parallel` is set as '`FALSE`', then the multiple simulations are run in series. We strongly recommend running multiple simulations in parallel, since this reduces computing time proportionally to the number of cpus used. Most computers today have dual or quad core processors, and many of them can handle hyper-threading (HT), which effectively splits one core into two virtual processors. This means that, with the average laptop with dual core and HT capabilities, one can potentially run up to 4 simulations in parallel.

To run 4 simulations in parallel on 4 cpus for a simple-shaped Gompertz model, the `basta()` function should be specified as:

```
> out <- basta(object = mydata, studyStart = 1990, studyEnd = 2000,
+     nsim = 4, parallel = TRUE, ncpus = 4)
```

4 Results

4.1 MCMC performance diagnostics

After the MCMC algorithms are finished, a range of diagnostics are calculated from the parameter chains. If multiple simulations were implemented and all of them have run through to completion, then potential scale reduction is calculated for each parameter to estimate convergence (Gelman *et al.* 2004). This diagnostic is calculated as $\hat{R} = \sqrt{\hat{v}^{+}/W}$, where W is a measure of the within-sequence variance and $\hat{v}^{+}$ is a weighted average of the between-sequence variance (B) and W. Convergence is attained when $\hat{R}$ is close to 1. As a rule of thumb, we have assigned an arbitrary upper bound of $\hat{R} < 1.1$ above which it is assumed that parameters have not reached convergence.

4.2 Model fit

If all parameters have converged, BaSTA calculates the deviance information criterion (DIC; Spiegelhalter *et al.* 2002), which has been described as a measure of predictive power and a criterion for model fit. DIC approximates the expected predictive deviance, and is calculated as;

$$\mathrm{DIC} = 2\hat{D}_{avg}(y) - D_{\hat{\theta}}(y)$$

where y denotes the observed data, $\hat{D}_{avg}(y)$ is the mean discrepancy between the data and the model as a function of the parameters θ, averaged over the posterior distribution, and $D_{\hat{\theta}}(y)$ is the discrepancy at the posterior mode (here represented by the point estimate $\hat{\theta}$). It is important to realise that the use of DICs is still controversial and, therefore, the results should to be taken with caution (see responses in Spiegelhalter *et al.* 2002). In order to improve the measure provided, BaSTA's DIC is calculated as an approximation of the group-marginalized DIC presented by Millar (2009).

4.3 Parameter comparison for categorical covariates

BaSTA also includes a diagnostic based on Kullback-Liebler discrepancies (KLD; Kullback & Leibler 1951, McCulloch 1989), that provides the user with a measure of how differently (or similarly) each categorical covariate affects survival. For instance, we may wish to evaluate the differences in survival between males and females with a simple Gompertz model, such that the mortality rate is of the form:

$$\mu(x|\boldsymbol{\theta}) = \exp\left[\overbrace{\left(\boldsymbol{\alpha}^T z\right)}^{b_0} + \overbrace{\left(\boldsymbol{\beta}^T z\right)}^{b_1} x\right], \tag{1}$$

where $\boldsymbol{\theta}$ are mortality parameters such that $\boldsymbol{\theta} = [b_0, b_1]$ and $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are subparameters that then both parameters b_0 and b_1 in equation 1 are evaluated as a function of these covariates. To illustrate the calculation of KLD, lets take b_0, for which the resulting 'sub-parameters' would be α_f and α_m such that, for an individual i, we have $b_0 = \alpha_f I_i + \alpha_m(1 - I_i)$, where I_i is an indicator function that assigns 1 if the individual is a female and 0 otherwise. For each of these parameters, BaSTA produces a posterior distribution, say $P_f = p(\alpha_f|\dots)$ and $P_m = p(\alpha_m|\dots)$, respectively. The KLD between these distributions is calculated as:

$$K(P_f, P_m) = \int_0^\infty P_f \log\left(\frac{P_f}{P_m}\right) d\alpha \tag{2}$$

The result can be interpreted as how far off we would be if we tried to predict α_m from the posterior distribution of α_f. If both distributions are identical, then $K(P_f, P_m) = 0$, suggesting that there is no distinction between males and females for b_0; as the KLD values increase the higher the discrepancy becomes. As can be inferred from equation 2, the relationship is asymmetric, namely $K(P_f, P_m) \neq K(P_m, P_f)$.

To make KLD easier to interpret McCulloch (1989) therefore proposed a simple calibration of the KLD values that reduces the asymmetry. This is as follows: Let $k = K(P_f, P_m)$ and $q(k)$ a calibration function such that

$$\begin{aligned} k &= K(P_f, P_m) \\ &= K(B(\frac{1}{2}), B(q(k))) \end{aligned}$$

where $B(\frac{1}{2})$ is a Bernouilli distribution for an event with probability 1/2 (i.e. same probability of success and failure). This calibration is then calculated as:

$$q(k) = \frac{(1 + (1 - e^{-2k})^{\frac{1}{2}})}{2} \tag{3}$$

Thus, $q(k)$ ranges from 0.5 to 1, where a value of 0.5 means that the distributions are identical, and 1 that there is no overlap between them.

4.4 Model outputs

The output provided by the `basta()` function is a list object of class `basta` that includes a range of diagnostics and results. This list includes summarized results in the form of coefficients (with standard errors and credible intervals), MCMC performance diagnostics, model settings as specified by the user, estimations of model fit and parameter overlap for categorical covariates, the raw traces from all runs, summarized values for times of birth and death, the data used in the model, and additional outputs such as life tables for each categorical covariate calculated from the estimated ages at death (without including left-truncated individuals).

5 Summarizing and plotting results

5.1 Printing results: functions `print()` and `summary()`

As we mentioned above, BaSTA's outputs can be explored using some of R's generic functions. For instance, the basic `print()` and `summary()` functions print a range of summary statistics and descriptions of the model used. Basic summary values can be visualized simply by typing the name of the BaSTA output object into the R console, while more information can be obtained with the `summary()` function. For example, here are the summary values for a simple-shaped Gompertz analysis on the simulated dataset included in the package, with 4 parallel simulations:

```
> summary(sim1Out, digits = 3)

Call:
Model                             : GO
Shape                             : simple
Covars. structure                 : fused
Cat. covars.                      : f, m
Cont. covars.                     :  weight

Model settings:
   niter   burnin thinning     nsim
   10000     5001      100        3

Runs:
All simulations finished.

Jumps and priors:
      Jump.sd Mean.priors
b0[f]   0.020       -3.00
```

```
b0[m]   0.020       -3.00
b1[f]   0.009        0.01
b1[m]   0.009        0.01
gamma   0.010        0.00

Mean Kullback-Liebler
discrepancy calibration (KLDC):
    b0    b1
f-m  1 0.974

Coefficients:
      Estimate  StdErr Lower95%CI Upper95%CI SerAutocor UpdateRate
b0[f]   -4.167 0.07863     -4.297     -4.029      0.751       0.12
b0[m]   -3.405 0.08151     -3.545     -3.253      0.706       0.12
b1[f]    0.177 0.00639      0.165      0.187      0.618       0.12
b1[m]    0.196 0.00878      0.179      0.211      0.579       0.12
gamma    0.208 0.02338      0.163      0.247      0.517       0.12
pi       0.624 0.00444      0.615      0.632     -0.111       1.00
      PotScaleReduc
b0[f]          1.03
b0[m]          1.08
b1[f]          1.04
b1[m]          1.07
gamma          1.01
pi             1.00

Convergence:
Appropriate convergence reached for all parameters.

DIC:
17573.7
```

5.2 Plotting results: function `plot()`

To visually verify that all parameter estimates have reached convergence, function `plot()` can be used on the BaSTA output object. Here is an example with the same output described above:

```
> plot(sim1Out)
```

This produces a plot of traces for the mortality parameters as in Fig. **??**:

In this case, no additional arguments are required. To plot the traces of the proportional hazards or recapture probability parameters or of the posterior chains, argument `trace.name` should be specified, with values `gamma`, `pi` or `post`. For instance, here is the code to plot the traces of the proportional hazards parameters:

```
> plot(sim1Out, trace.name = "gamma")
```

In addition, the predicted survival probabilities and mortality rate functions for the different categorical covariates can be plotted by typing:

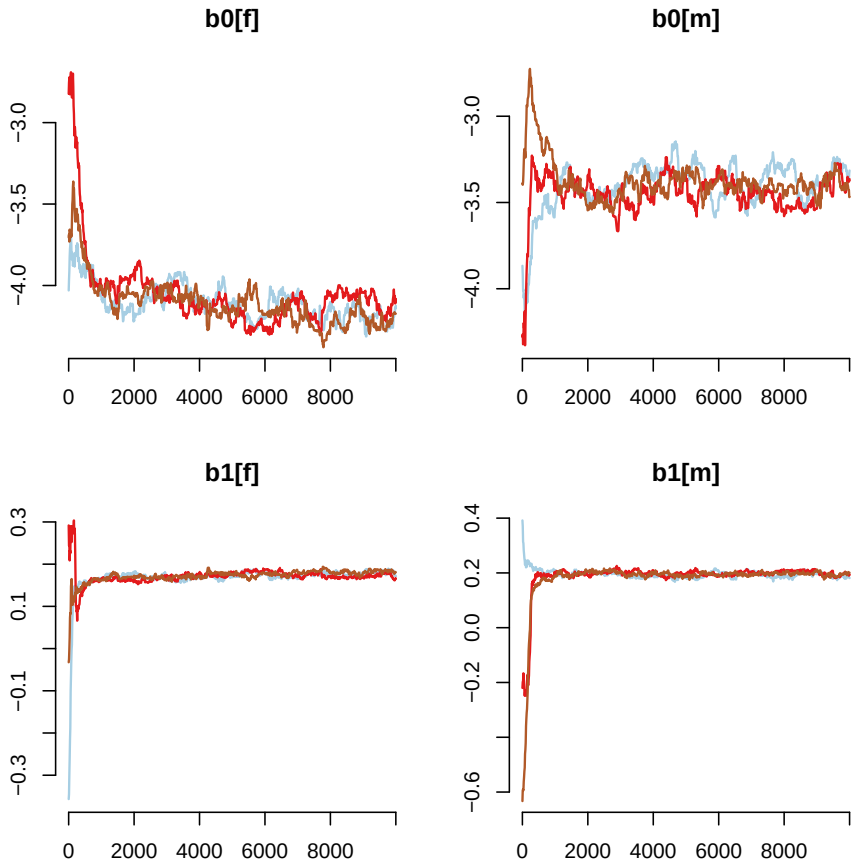

Figure 3: Traces for the model parameters.

```
> plot(sim1Out, plot.trace = FALSE)
```

which produces the following plot (Fig. 4):

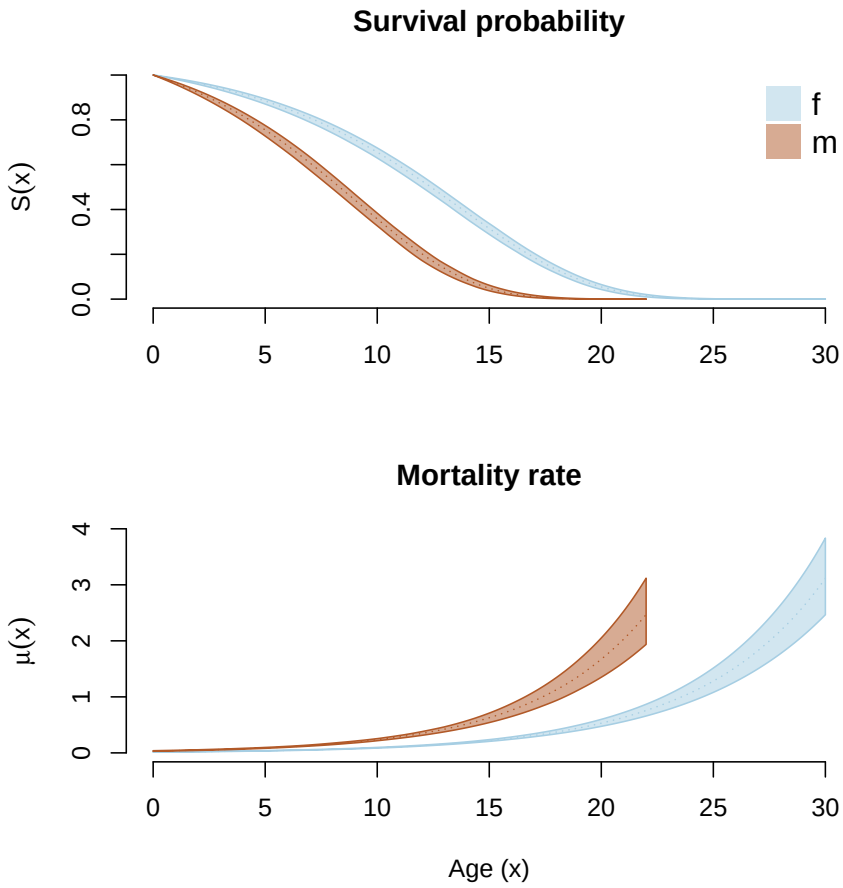

Figure 4: Survival and mortality trajectories from the example data analysis.

References

Chambers, J. & Hastie, T. (1992) *Statistical Models in S.* Wadsworth & Brooks/Cole.

Clark, J.S. (2007) *Models for ecological data.* Princeton University Press, Princeton, New Jersey.

Colchero, F. & Clark, J.S. (2011) Bayesian inference on age-specific survival for censored and truncated data. *Journal of Animal Ecology.*

Colchero, F., Jones, O.R. & Rebke, M. (2011) BaSTA: an R package for Bayesian estimation of age-specific mortality from incomplete mark-recapture/recovery data with covariates. in prep.

Cox, D.R. & Oakes, D. (1984) *Analysis of Survival Data.* Chapman and Hall, London, UK.

Gelman, A., Carlin, J., Stern, H. & Rubin, D. (2004) *Bayesian data analysis*, chap. 11, pp. 283–310. Chapman & Hall/CRC, Washington, DC, 2nd edn.

Gompertz, B. (1825) On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philosophical Transactions of the Royal Society of London*, **115**, 513–583.

Klein, J. & Moeschberger, M. (2003) *Survival analysis. Techniques for censored and truncated data.* Springer, New York, New York, 2nd edn.

Knaus, J. (2010) *snowfall: Easier cluster computing (based on snow).* R package version 1.84. URL `http://CRAN.R-project.org/package=snowfall`

Kullback, S. & Leibler, R.A. (1951) On information and sufficiency. *Annals of Mathematical Statistics*, **22**, 79–86.

McCulloch, R.E. (1989) Local model influence. *Journal of the American Statistical Association*, **84**, 473–478.

Millar, R.B. (2009) Comparison of hierarchical bayesian models for overdispersed count data using DIC and Bayes' factors. *Biometrics*, **65**, 962–969.

Pinder III, J.E., Wiener, J.G. & Smith, M.H. (1978) The Weibull distribution: a method of summarazing survivorship data. *Ecology*, **59**, 175–179.

Pletcher, S. (1999) Model fitting and hypothesis testing for age-specific mortality data. *Journal of Evolutionary Biology*, **12**, 430–439.

R Development Core Team (2011) *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. URL `http://www.R-project.org/`

Siler, W. (1979) A competing-risk model for animal mortality. *Ecology*, **60**, 750–757.

Spiegelhalter, D., Best, N., Carlin, B. & Linde, A.V.D. (2002) Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**, 583–639.

White, G. & Burnham, K. (1999) Program MARK: survival estimation from populations of marked animals. *Bird Study*, **46**, 120–139.

6 Additional information

6.1 Mortality model details

Table 4: Mortality rates for all models included in BaSTA.

Function	**Mortality rate** $\mu_b(x\|\mathbf{b})$	**Survival probability** $S_b(x\|\mathbf{b})$	**Parameters**
Exponential	b	e^{-bx}	$b > 0$
Gompertz	$e^{b_0+b_1 x}$	$\exp\left[\frac{e^{b_0}}{b_1}(1-e^{b_1 x})\right]$	$-\infty < b_0, b_1 < \infty$
Weibull	$b_0 b_1^{b_0} x^{b_0-1}$	$\exp\left[-(b_1 x)^{b_0}\right]$	$b_0, b_1 > 0$
Logistic	$\frac{e^{b_0+b_1 x}}{1+b_2\frac{e^{b_0}}{b_1}(e^{b_1 x}-1)}$	$\left(1+b_2\frac{e^{b_0}}{b_1}\left(e^{b_1 x}-1\right)\right)^{-1/b_2}$	$b_0, b_1, b_2 > 0$