BaSTA

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

Fernando Colchero
Department of Mathematics and Computer Science
Interdisciplinary Center on Population Dynamics
University of Southern Denmark

Contents

1 Introduction **2**

2 Step-by-step tutorial **3**

2.1 Data formatting . . . 3
2.1.1 Capture-mark-recapture data . . . 3
2.1.2 Census data . . . 6
2.2 Verifying data consistency: function **DataCheck()** . . . 6
2.2.1 Capture-mark-recapture data . . . 6
2.2.2 Census data . . . 7
2.3 Setting up the analysis: function **basta()** . . . 9
2.4 Choosing mortality models: arguments **model** and **shape** . . . 9
2.4.1 Conditioning on a minimum age: the **minAge** argument . . . 11
2.4.2 Including covariates . . . 11
2.4.3 Recapture probabilities for CMR data . . . 12
2.4.4 MCMC general settings: the **niter**, **burnin** and **thinning** arguments . . . 12
2.4.5 Initial parameters, jumps and priors . . . 12
2.4.6 Updating jumps: argument **updateJumps** . . . 13
2.4.7 Multiple runs: arguments **nsim**, **parallel** and **ncpus** . . . 14
2.5 Results . . . 15
2.5.1 Model outputs . . . 15
2.5.2 Printing results: functions **print()** and **summary()** . . . 15
2.5.3 Plotting results: function **plot()** . . . 16

3 Technical details **16**

3.1 Survival analysis and mortality models included in BaSTA . . . 16
3.2 General properties of the mortality models . . . 19
3.3 Broken stick model for survival after a minimum age . . . 21
3.4 Covariates in mortality models . . . 22

3.5 MCMC convergence diagnostic 22
3.6 Model fit 22
3.7 Parameter comparison for categorical covariates 22

References **24**

1 Introduction

Here we present BaSTA (Bayesian Survival Trajectory Analysis; Colchero *et al.* (2012)), a free open-source R package (R Development Core Team 2011) that implements the hierarchical Bayesian model described by Colchero & Clark (2012). 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. To cite this package, please refer to the paper by Colchero, Jones & Rebke (2012) published in Methods in Ecology and Evolution. This document is based on the Supporting Information (appendix) published with that paper.

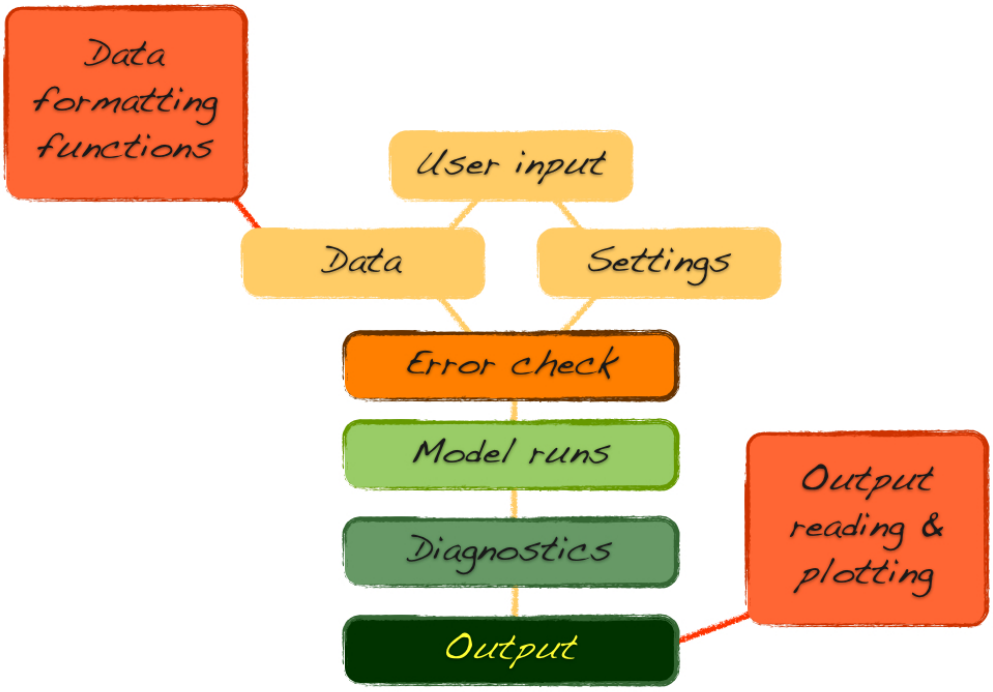

Figure 1: Work flow of BaSTA's main function **`basta()`** after data input and argument definition from the user. During the initial "Error check" sequence, 1 implies that no errors were detected and 0 means otherwise. In the later case, the function is stopped and an error message is printed explaining 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. 1). 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 2012). 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 a data formatting function, **CensusToCaptHist()** and a data checking function **DataCheck()**.

2 Step-by-step tutorial

2.1 Data formatting

2.1.1 Capture-mark-recapture data

BaSTA's capture-mark-recapture (CMR) input data format is compatible with other programs that carry out analysis data sets such as MARK (White & Burnham 1999). The data need 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. The years of birth and death are not coded as detection. If covariates are to be included, additional columns can be added, with one column per covariate. In the following sections we show how to use BaSTA's built-in functions to organize the input data.

Constructing the capture history matrix: CensusToCaptHist()

Commonly, capture-recapture datasets are stored in what we call "census" or "survey" tables. Typically, these tables have one row for each time an individual is observed, and include one column for individual IDs and one column for detection date. Often, these tables are stored in a spreadsheet software (e.g. Excel). We recommend that users save it either in tab delimited text (.txt) or comma separated value (.csv) formats so they can be easily read into R. Below is the code to read a table that was previously saved in Excel as tab delimited text:

```
censusMat <- read.table("path to census table.txt",
                        sep = "\t", header = TRUE)
```

If the table was saved in .csv format, it can be read as:

```
censusMat <- read.csv("path to census table.csv", header = TRUE)
```

Here is an example of a simulated dataset in a table called **censusMat** which includes five individuals captured between 1990 and 1995. We can use the command **head()** to look at the first few rows of data:

```
head(censusMat)

>      ID Year
> [1,]  1 1990
> [2,]  1 1992
> [3,]  1 1993
> [4,]  2 1991
> [5,]  2 1992
> [6,]  2 1994
```

This data can then be used as the input for the **CensusToCaptHist()** function in order to construct the recapture matrix required by BaSTA:

```
Y <- CensusToCaptHist(ID = censusMat[, 1], d = censusMat[, 2])
```

The **ID** argument in the **CensusToCaptHist()** function, takes a vector with the individual IDs, in this case the first column in **censusMat**, along with the **d** argument, which is a vector of dates on which each individual was detected (i.e. the second column in **censusMat**), and argument **dformat** which is (optionally) used to specify the date format used in **d**. In case the **d** argument is a character string based on a date format, or of class **date**, the minimun time interval can be specified with argument **timeInt**, which takes a single character value, with the folowing options: "**Y**" for years (default); "**M**" for months; "**W**" for weeks; and "**D**" for days.

The resulting capture history matrix (**Y**) looks like this:

```
>   ID X1990 X1991 X1992 X1993 X1994 X1995
> 1  1     1     0     1     1     0     0
> 2  2     0     1     1     0     1     0
> 3  3     0     0     1     1     0     0
> 4  4     0     1     1     0     0     1
> 5  5     0     0     0     1     0     1
```

Matrix for times of birth and death In addition, a two column matrix for the years of birth and death needs to be appended to the capture-history matrix. Once again, this matrix can be read from any spreadsheet format after being saved in .txt or .csv format (see example above). This matrix should contain the years of birth and death when these are known, or **NA** otherwise. For example, below we show a hypothetical matrix called **birthDeath** for the five individuals in the preceding example:

```
print(birthDeath)

>      ID Birth Death
> [1,]  1    NA  1995
> [2,]  2  1990    NA
> [3,]  3  1991  1994
> [4,]  4    NA    NA
> [5,]  5    NA    NA
```

In this case, the years of birth for individuals 1, 4, and 5 are unknown, while years of death are only known for individuals 1 and 3.

Additional columns for minimum and maximum times of birth and/or death can be included in case unknown times of birth and/or death can be bound. These columns should be included after the columns corresponding to the recapture matrix (see following section).

Merging the capture history, birth-death and covariate matrices

A matrix or data frame of covariates (i.e., explanatory variables) should include any relevant predictors you want to test and it should be organized in the same order as the birth-death matrix (i.e., sharing the same IDs). For instance, here is a hypothetical example including weight at birth and sex:

```
print(covMat)

>   ID sex weight
> 1  1   f    9.5
> 2  2   f   10.8
> 3  3   m   12.3
> 4  4   f    8.6
> 5  5   m   11.9
```

If the order of the rows in the birth-death, the capture-history and the covariate matrices is the same (i.e. each row corresponds to the same individual), then these three matrices or data frames can be merged very simply as follows:

```
inputDat <- data.frame(birthDeath, Y[, -1], covMat[, -1])
print(inputDat)

>   ID Birth Death X1990 X1991 X1992 X1993 X1994 X1995 sex weight
> 1  1    NA  1995     1     0     1     1     0     0   f    9.5
> 2  2  1990    NA     0     1     1     0     1     0   f   10.8
> 3  3  1991  1994     0     0     1     1     0     0   m   12.3
> 4  4    NA    NA     0     1     1     0     0     1   f    8.6
> 5  5    NA    NA     0     0     0     1     0     1   m   11.9
```

In this case, it is important to remove the first column (i.e., **`[,-1]`**) of the **`Y`** and **`covMat`** matrices in case they still include the IDs column.

If minimum and maximum times of birth and/or death are included, for instance see example below:

```
print(minMax)

>   ID Min.Birth Max.Birth Min.Death Max.Death
> 1  1      1985      1989        NA        NA
> 2  2        NA        NA      1995      1997
> 3  3        NA        NA        NA        NA
> 4  4      1987      1990      1996      1999
> 5  5      1990      1992      1995      1998
```

then this additional matrix or data frame should be included after the capture-recapture matrix as:

```
inputDatCMR <- data.frame(birthDeath, Y[, -1], minMax[, -1], covMat[, -1])
print(inputDatCMR)

>   ID Birth Death X1990 X1991 X1992 X1993 X1994 X1995 Min.Birth
> 1  1    NA  1995     1     0     1     1     0     0      1985
> 2  2  1990    NA     0     1     1     0     1     0        NA
> 3  3  1991  1994     0     0     1     1     0     0        NA
> 4  4    NA    NA     0     1     1     0     0     1      1987
> 5  5    NA    NA     0     0     0     1     0     1      1990
>   Max.Birth Min.Death Max.Death sex weight
> 1      1989        NA        NA   f    9.5
> 2        NA      1995      1997   f   10.8
> 3        NA        NA        NA   m   12.3
> 4      1990      1996      1999   f    8.6
> 5      1992      1995      1998   m   11.9
```

Importantly, the column names for the minimum and maximum birth rates need to be precisely as shown above, otherwise the **`DataCheck()`** function will not recognize them.

If the order of the rows in these matrices is *not* the same, then function **`merge`** should be used (twice) to collate all three matrices as follows:

```
inputDatCMR <- merge(birthDeath, Y, by.x = "ID", by.y = "ID")
inputDatCMR <- merge(inputDatCMR, minMax, by.x = "ID", by.y = "ID")
inputDatCMR <- merge(inputDatCMR, covMat, by.x = "ID", by.y = "ID")
```

The **`merge`** function must be used three times because three matrices with different row order need to be combined using the column **`ID`** as the reference.

2.1.2 Census data

For data sets of type **"census"**, the input data frame needs to include the at least the following columns, with exactly these column names:

```
> [1] "ID"             "Birth.Date"     "Min.Birth.Date"
> [4] "Max.Birth.Date" "Entry.Date"     "Depart.Date"
> [7] "Depart.Type"
```

All the data columns need to be formatted as **YYYY-mm-dd**. The **Depart.Date** column should be coded as **"C"** for censored (i.e., still alive at the **Depart.Date**) and **"D"** if the **Depart.Date** is the death date. Here are the first six rows of the example data built-in to BaSTA:

```
>   ID Birth.Date Min.Birth.Date Max.Birth.Date Entry.Date Depart.Date
> 1  1 1995-04-02     1995-04-02     1995-04-02 1995-04-02  1998-11-14
> 2  2 1992-11-15     1991-11-15     1993-01-03 1993-01-03  1994-12-05
> 3  3 1994-08-04     1994-08-04     1994-08-04 1994-08-04  1999-05-23
> 4  4 1993-10-01     1992-01-01     1993-12-09 1993-12-09  1995-12-31
> 5  5 1990-09-10     1990-09-10     1990-09-10 1990-09-10  1998-06-14
>   Depart.Type
> 1           C
> 2           D
> 3           D
> 4           C
> 5           D
```

Note that rows 2 and 4 have columns **Min.Birth.Date** and **Max.Birth.Date** with different dates from **Birth.Date**. These indicate that the birth date for these records is not known precicely and thus requires minimum and maximum bounds. When the birth dates are known exactly, then **Min.Birth.Date** and **Max.Birth.Date** should have the same date as **Birth.Date**, as in rows 1, 3, and 5.

In addition, covariates can be included, for instance as in the example below:

```
>   ID Birth.Date Min.Birth.Date Max.Birth.Date Entry.Date Depart.Date
> 1  1 1995-04-02     1995-04-02     1995-04-02 1995-04-02  1998-11-14
> 2  2 1992-11-15     1991-11-15     1993-01-03 1993-01-03  1994-12-05
> 3  3 1994-08-04     1994-08-04     1994-08-04 1994-08-04  1999-05-23
> 4  4 1993-10-01     1992-01-01     1993-12-09 1993-12-09  1995-12-31
> 5  5 1990-09-10     1990-09-10     1990-09-10 1990-09-10  1998-06-14
>   Depart.Type sex weight
> 1           C   f    9.5
> 2           D   f   10.8
> 3           D   m   12.3
> 4           C   f    8.6
> 5           D   m   11.9
```

2.2 Verifying data consistency: function DataCheck()

2.2.1 Capture-mark-recapture data

After the final CMR data frame is constructed, its consistency can be verified with the **DataCheck()** function. This function performs a range of diagnostic checks on the data frame (Table 3). Below is an example with the simulated dataset described above:

Arguments **studyStart** and **studyEnd** correspond to the years of start and end of the study. If the **silent** argument is set as **FALSE** or if you use function **print(checkedData)**, the following range of descriptive statistics about the dataset is printed to the console:

```
> DATA CHECK: 2022-10-02 12:27:08
> ================================
>
> DATA SUMMARY:
> =============
> - Number of individuals:           5
> - Number with known birth year:    2
> - Number with known death year:    2
> - Number with known birth
>   AND death years:                 1
> - Number of recaptures:           13
> - Earliest detection year:      1990
> - Latest detection year:        1995
> - Earliest birth year:          1990
> - Latest birth year:            1991
> - Earliest death year:          1994
> - Latest death year:            1995
>
> NUMBER OF PROBLEMATIC RECORDS:
> ==============================
> No problematic records detected.
```

DataCheck() searches the data set for six different types of error (Table 3). The output object from function **DataCheck()** (in this example object **checkedData**) includes a list with the row numbers of each record where errors were detected. This allows the user to return to the original data and, if necessary, manually correct the records with issues. If argument **silent = FALSE** and an error is detected, function **DataCheck()** prints to the console a message informing the user of the errors found. For instance, if the third record in **inputDat** is modified as:

```
>   ID Birth Death X1990 X1991 X1992 X1993 X1994 X1995 sex weight
> 3  3  1991  1990     0     0     1     1     0     0   m   12.3
```

then, when evaluated with **DataCheck**, the function prints the folowing error message:

```
dataWithError <- DataCheck(inputDatErr, studyStart = 1990,
                     studyEnd = 1995, silent = FALSE)

> The following rows have birth dates that are later than their death dates:
> [1] 3
> The following rows have observations that occur after the year of death:
> [1] 3
> Problems were detected with the data
> You can use function FixCMRdata() to resolve issues.
```

In case any of these issues are detected, it is possible to correct them by means of function **FixCMRdata()** with the following code:

```
fixedDataCMR <- FixCMRdata(object = inputDatCMR, studyStart = 1990,
                           studyEnd = 1995, autofix = rep(1, 6))
```

Note that argument **autofix** controls the type of data correction chosen as shown in Table 3. 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 fix them.

2.2.2 Census data

If the data is of type **"census"**, the function **DataCheck()** verifies inconsistencies between the date columns. Thus, to check the census type input data only type:

Table 1: 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

```
checkedDataCens <- DataCheck(object = inputDatCens, dataType = "census",
                             silent = FALSE)

>
> No inconsistencies between dates.
```

The output object can be printed to the screen, which produces the following summary:

```
print(checkedDataCens)

> DATA CHECK: 2022-10-02 12:27:08
> ================================
>
> DATA SUMMARY:
> =============
> Total number of records  : 5
> Number censored (C)      : 2
> Number uncensored (D)    : 3
> Number with unknown birth: 2
>
> NAs IN DATES COLUMNS:
> ---------------------
> No NAs found in dates columns.
>
> DATES RANGES:
> -------------
> Birth.Date Min.Birth.Date Max.Birth.Date Entry.Date Depart.Date
> 1990-09-10 1990-09-10 1990-09-10 1990-09-10 1994-12-05
> 1995-04-02 1995-04-02 1995-04-02 1995-04-02 1999-05-23
>
> INCONSISTENCIES BETWEEN DATES COLUMNS:
> --------------------------------------
> None.
>
> INCONSISTENCIES IN DEPARTURE TYPES:
```

```
> ----------------------------------
> None
```

If inconsistencies are found, then the function produces a warning message indicating that there are inconsistencies with the data. For instance, if the time of birth for the second individual had a typo on the **Birth.Date** such that the birth date is specified as **"1993-11-15"**, then **DataCheck** produces the following message:

```
>
> The following number of records
> have inconsistencies between dates:
> Birth > Max Birth      : 1 records
> Birth > Entry          : 1 records
>
> Use print(object) to find inconsistent records.
```

The output object consists of a list for each of the potential errors, each of which including a vector with the row numbers where the errors were found.

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

After the data have been formatted and verified for consistency, the analysis can be performed with the **basta()** function. In this section we explain the arguments used in this function and we provide background information on the models used in BaSTA. 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 the simulated dataset of type **CMR** described above (i.e. **inputDat**) can be performed by entering the following command:

```
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995)
```

Note that if the input data is of type **census**, then the call needs to be modified as

```
out <- basta(object = inputDatCens, dataType = "census")
```

Note that arguments **studyStart** and **studyEnd** are not necessary for census data, with default value **NULL**.

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 by typing **?basta** in the R console. 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 (if they have covariates). Below we provide a description of the mortality models implemented in BaSTA and we outline how to set up an analysis to test a range of models and covariate data structures.

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

The **model** argument can be used to choose between four basic mortality 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). Each one of these functions can describe different trends in age-specific mortality, giving BaSTA considerable flexibility when estimating these vital rates (Fig. 2). For a detailed explanation on the properties of these models please refer to section 'Technical details' in this document.

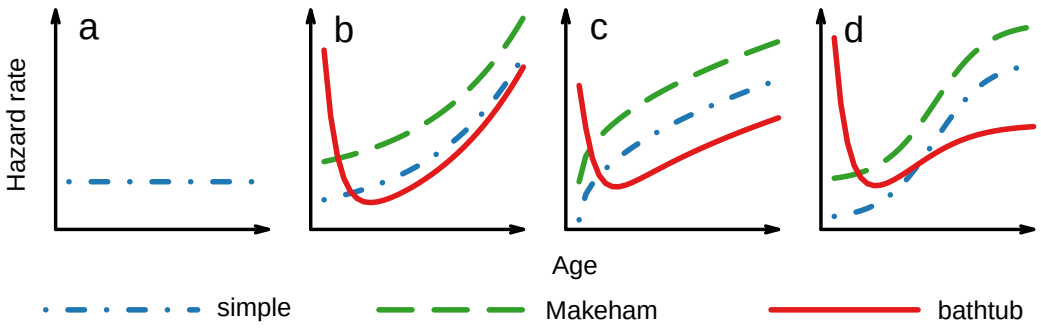

Figure 2: Mortality rates, $\mu(x|\theta)$, resulting from the four basic models included in BaSTA: a) exponential; b) Gompertz; c) Weibull; and d) logistic. The three different lines in each plot (except in a) show examples of the shapes that can be tested with BaSTA, namely: 'simple'; 'Makeham'; and 'bathtub'. Modified from Colchero, Jones & Rebke (2012).

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 **??**; 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. 2. Clearly, the number of parameters used in each of these combinations varies. In table 2 we show the number of parameters for the different types of mortality models and shape combinations.

Table 2: 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:

```
# Analysis for CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             model = "LO", shape = "bathtub")

# Analysis for census data:
out <- basta(object = inputDatCens, dataType = "census",
             model = "LO", shape = "bathtub")
```

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

2.4.1 Conditioning on a minimum age: the `minAge` argument

In some species, the fates of individuals younger than a certain age are typically unknown. For example, after fledging, juvenile seabirds disperse for several years, during which they can experience high mortality. After this time, they may settle in a different colony from the colony where they were born, and are thus never detected again. Consequently, uncertainty in the fate of juveniles can inflate early mortality estimates. Accordingly, BaSTA allows users to condition the analysis to survival after a minimum age with argument **`minAge`**. For example, to evaluate a simple Gompertz model on individuals older than 2, the code should be:

```
# Analysis for CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             minAge = 2)

# Analysis for census data:
out <- basta(object = inputDatCens, dataType = "census",
             minAge = 2)
```

2.4.2 Including covariates

To include covariates into the analysis, then argument **`formulaMort`** should be specified following the same convention as any other regression like function. Here is an example with **`Sex`** as a covariate:

```
# Analysis for CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             minAge = 2, formulaMort = ~ Sex - 1)

# Analysis for census data:
out <- basta(object = inputDatCens, dataType = "census",
             minAge = 2, formulaMort = ~ Sex - 1)
```

BaSTA also allows users to define three optional structures to evaluate the effect of covariates on age patterns of survival with argument **`covarsStruct`**: i) **`fused`** (default), in which covariates are separated into continuous and categorical types, where the former are included into a proportional hazards framework (Klein & Moeschberger 2003), and the latter are included as linear functions of the survival parameters, in an analogous manner to the way they are handled in generalised linear models (GLMs); ii) **`prop.haz`**, where all covariates are included under a proportional hazards structure; and iii) *all.in.mort*, where all covariates are evaluated as linear functions of the survival parameters. The latter structure is currently only implemented with a simple-shaped Gompertz model. In the proportional hazards model, differences between parameter estimates (e.g. differences between the parameter for two sexes) are expressed as *gamma* parameters.

Figure 3 shows a schematic representation of the potential effects that can be depicted based on the choice of covariate structure with one categorical covariate (e.g. sex) and one continuous covariate (e.g. weight) using a Gompertz model.

For instance, the specifications for an analysis with all covariates included in the mortality functions should be:

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             covarsStruct = "all.in.mort")

# For census data:
out <- basta(object = inputDatCens, dataType = "census",
             covarsStruct = "all.in.mort")
```

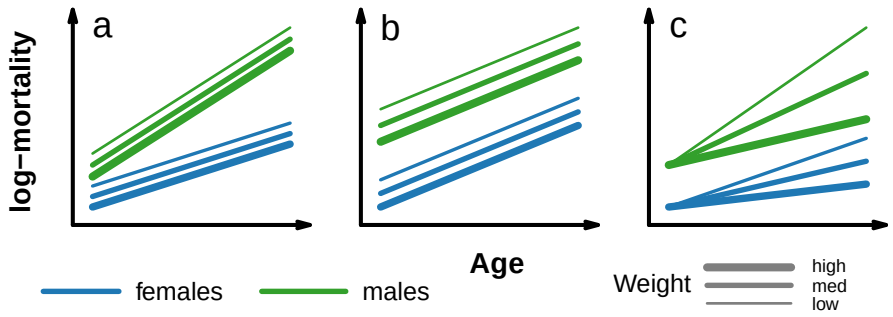

Figure 3: Simulated log-mortality as a function of sex and weight based on the choice of covariate structure: a) **fused**; b) **prop.haz**; and c) **all.in.mort**.

2.4.3 Recapture probabilities for CMR data

Currently BaSTA allows the user to test if recapture probability should be constant (default) or if it should be time dependent. This could be useful when the recapture effort is not sustained equally in time. For example, to specify that recapture probabilities changed after year 1993 for our simulated data set above, argument **recaptTrans** needs a two element vector with the first element being the first year of the study (in this case 1990) and the second element the year when recapture probabilities changed (i.e. 1993). Thus, the model can be specified as:

```
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             recaptTrans = c(1990, 1993))
```

2.4.4 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 (2012), 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. Thus, for a short run of 1,000 iterations with a burnin of 100 steps and a thinned sequence every 10 steps, the model can be specified as:

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             niter = 1000, burnin = 100, thinning = 10)

# For census data:
out <- basta(object = inputDatCens, dataType = "census",
             niter = 1000, burnin = 100, thinning = 10)
```

2.4.5 Initial parameters, jumps and priors

Initial parameters are simply the starting point for the parameter chains. Then, the jump values define the standard deviation of the distribution from which BaSTA's MCMC algorithm draws the next value in the chain. Larger values will result in a faster traversal of parameter space. If the values are too small parameter

space is not efficiently searched, but if the values are too high there is a risk that the optimal values might be missed.

Prior values reflect our belief in the values of the parameters. Ideally, the values selected for the priors will have no strong effect on the outcome of the analysis (i.e. the posterior parameter estimates).

Although BaSTA has default values for these initial parameters, jump standard deviations, and priors, they can be modified with the arguments **thetaStart** and **gammaStart** for mortality and proportional hazards initial parameters, and the corresponding **thetaJumps**, **thetaPriorMean**, **thetaPriorSd**, **gammaJumps**, **gammaPriorMean** and **gammaPriorSd** 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 2) and a '**mixed**' covariate structure is chosen, and two categorical and two continuous covariates are included in the dataset, **thetaStart**, **thetaJumps**, **thetaPriorMean** and **thetaPriorSd** should be vectors of length 3 (if we want the same set of parameters for both categorical covariates), or of length 6 (if we want a single set of parameters per covariate), or matrices of dimension 2×3. Also, **gammaStart**, **gammaJumps**, **gammaPriorMean** and **gammaPriorSd**, 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:

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990,
             studyEnd = 2000, model = "LO",
             shape = "simple", thetaJumps = c(0.1, 0.1, 0.1))

# For census data:
out <- basta(object = inputDatCens, dataType = "census", model = "LO",
             shape = "simple", thetaJumps = c(0.1, 0.1, 0.1))
```

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

```
newJumps <- 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:

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             model = "LO", shape = "simple", thetaJumps = newJumps)

# For census data:
out <- basta(object = inputDatCens, dataType = "census",
             model = "LO", shape = "simple", thetaJumps = newJumps)
```

2.4.6 Updating jumps: argument updateJumps

A difficult issue with hierarchical models that use Metropolis algorithms to sample parameters is to find the appropriate combination of jump standard deviations for these parameters. BaSTA uses this type of

sampling scheme for all mortality parameters. BaSTA allows the user to ask the algorithm to find the right combination of jump sd's without having to manually run it several times before finding the jumps. This can be achieved by setting argument **updateJumps** as **TRUE**. If argument **updateJumps** is **TRUE**, then BaSTA runs an initial simulation only to find the appropriate combination of jumps. This initial run implements an Adaptive Metropolis algorithm as proposed by Roberts & Rosenthal (2009) which runs for 10,000 steps after which it starts the main simulation(s) that will constitute the final results of the analysis. For example, to set this routine on our hypothetical dataset and to run the model for two simulations in parallel, the code can be specified as folows:

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             model = "LO", shape = "simple", nsim = 2,
             parallel = TRUE, ncpus = 2, updateJumps = TRUE)

# For census data:
out <- basta(object = inputDatCens, dataType = "census",
             model = "LO", shape = "simple", nsim = 2,
             parallel = TRUE, ncpus = 2, updateJumps = TRUE)
```

As you can note, it is not necessary to specify jumps, however, if the user has a general idea of where the jumps should lie, then they can be specified. This will only change the initial values where the jumps are tested. If, on the other hand, only one simulation has been specified (i.e. argument **nsim** = 1) then BaSTA runs the update jump routine on the main simulation. This can be useful in case the user wants to find appropriate jumps manually.

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

```
# For CMR data:
out <- basta(object = inputDatCMR, studyStart = 1990, studyEnd = 1995,
             nsim = 4, parallel = TRUE, ncpus = 4)

# For census data:
out <- basta(object = inputDatCens, dataType = "census",
             nsim = 4, parallel = TRUE, ncpus = 4)
```

2.5 Results

2.5.1 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), the raw traces from all runs, summarized values for times of birth and death, and MCMC performance diagnostics such as convergence, parameter update rates and serial autocorrelation within each parameter chain. General information on model settings as specified by the user is also included, as well as estimates of model fit and parameter overlap for categorical covariates. In addition, the data used in the model and optional outputs such as life tables for each categorical covariate calculated from the estimated ages at death are also included.

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

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 models used for inference. 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(bastaCMRout, digits = 3)

>
> Call:
> Model              : GO
> Shape              : simple
> Covars. structure  : 0
> Minimum age        : fused
> Cat. covars.       : SexFemale, SexMale
> Cont. covars.      :  Weight
>
> Model settings:
>    niter   burnin thinning     nsim
>    55000     5001       50        4
>
> Mean Kullback-Leibler
> discrepancy calibration (KLDC):
>                        b0    b1
> SexMale - SexFemale 0.991 0.979
>
> Coefficients:
>                Mean StdErr Lower95%CI Upper95%CI SerAutocorr
> b0.SexFemale -3.590 0.2139    -4.0235     -3.175      0.6202
> b0.SexMale   -3.026 0.1743    -3.3740     -2.691      0.3933
> b1.SexFemale  0.133 0.0204     0.0936      0.174      0.7232
> b1.SexMale    0.186 0.0214     0.1431      0.227      0.4136
> gamma.Weight  0.368 0.0700     0.2330      0.506      0.3168
> pi            0.612 0.0102     0.5916      0.632      0.0537
>              UpdateRate PotScaleReduc
> b0.SexFemale      0.249          1.01
> b0.SexMale        0.254          1.00
> b1.SexFemale      0.236          1.00
> b1.SexMale        0.244          1.00
```

```
> gamma.Weight      0.253          1.00
> pi                1.000          1.00
>
> Convergence:
> Appropriate convergence reached for all parameters.
>
> DIC:
> 3220.284
```

2.5.3 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(bastaCMRout)
```

This produces a plot of traces for the mortality parameters as in Fig. 2.5.3.

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(bastaCMRout, trace.name = "gamma")
```

In addition, the predicted survival probabilities and mortality rate functions for the different categorical covariates can be plotted by typing modifying the **`plot.type`** argument as follows:

```
plot(bastaCMRout, plot.type = "demorates")
```

which produces Fig. 5.

3 Technical details

3.1 Survival analysis and mortality models included in BaSTA

BaSTA uses the framework described by Colchero & Clark (2012), in which age-specific survival trends and the latent (i.e. unknown) times of birth and death are estimated using parametric functions for mortality (commonly expressed as the hazard rate) and survival probability. This hazard rate or mortality is usually expressed as:

$$\mu(x|\boldsymbol{\theta}) = \lim_{\Delta x \to 0} \frac{\Pr\left(x < X < x + \Delta x | X > x, \boldsymbol{\theta}\right)}{\Delta x}, \tag{1a}$$

where x is age and X is a random variable for ages at death, while $\boldsymbol{\theta}$ is the vector of survival parameters to be estimated. From equation 1a, the following functions can be derived:

$$S(x|\boldsymbol{\theta}) = \Pr\left(X > x\right) = \exp\left[-\int_0^x \mu(y|\boldsymbol{\theta})dy\right], \tag{1b}$$

$$F(x|\boldsymbol{\theta}) = \Pr\left(X < x\right) = 1 - S(x|\boldsymbol{\theta}), \tag{1c}$$

$$f(x|\boldsymbol{\theta}) = \Pr\left(x < X < x + dx\right) = \mu(x|\boldsymbol{\theta})S(x|\boldsymbol{\theta}), \tag{1d}$$

where equation (1b) is the survival probability, (1c) is the the probability that death occurs before age x (or cumulative density function, cdf), and (1d) is the probability density function (pdf) of ages at death.

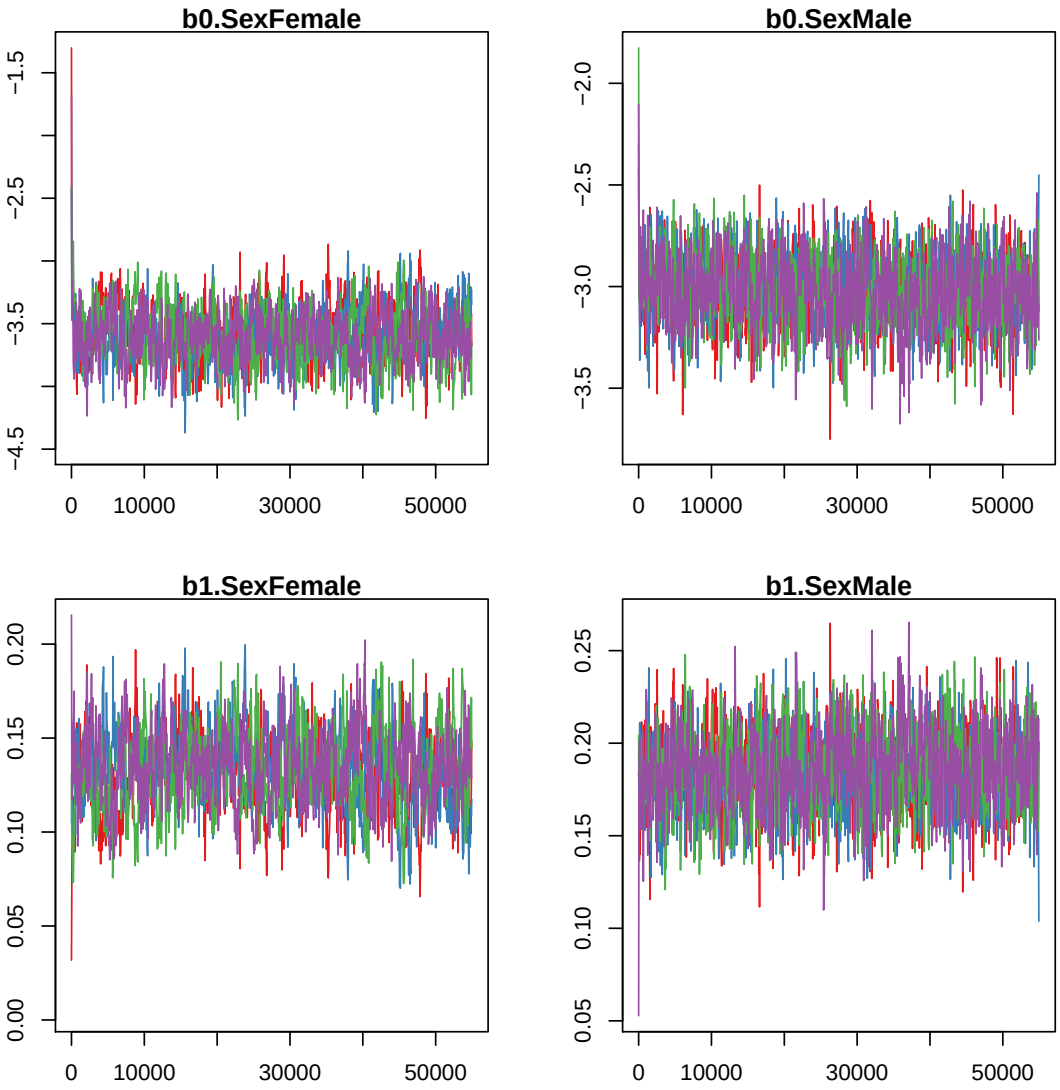

Figure 4: Traces for parameters b_0 and b_1 on a simple Gompertz model with 4 parallel runs and sex covariate.

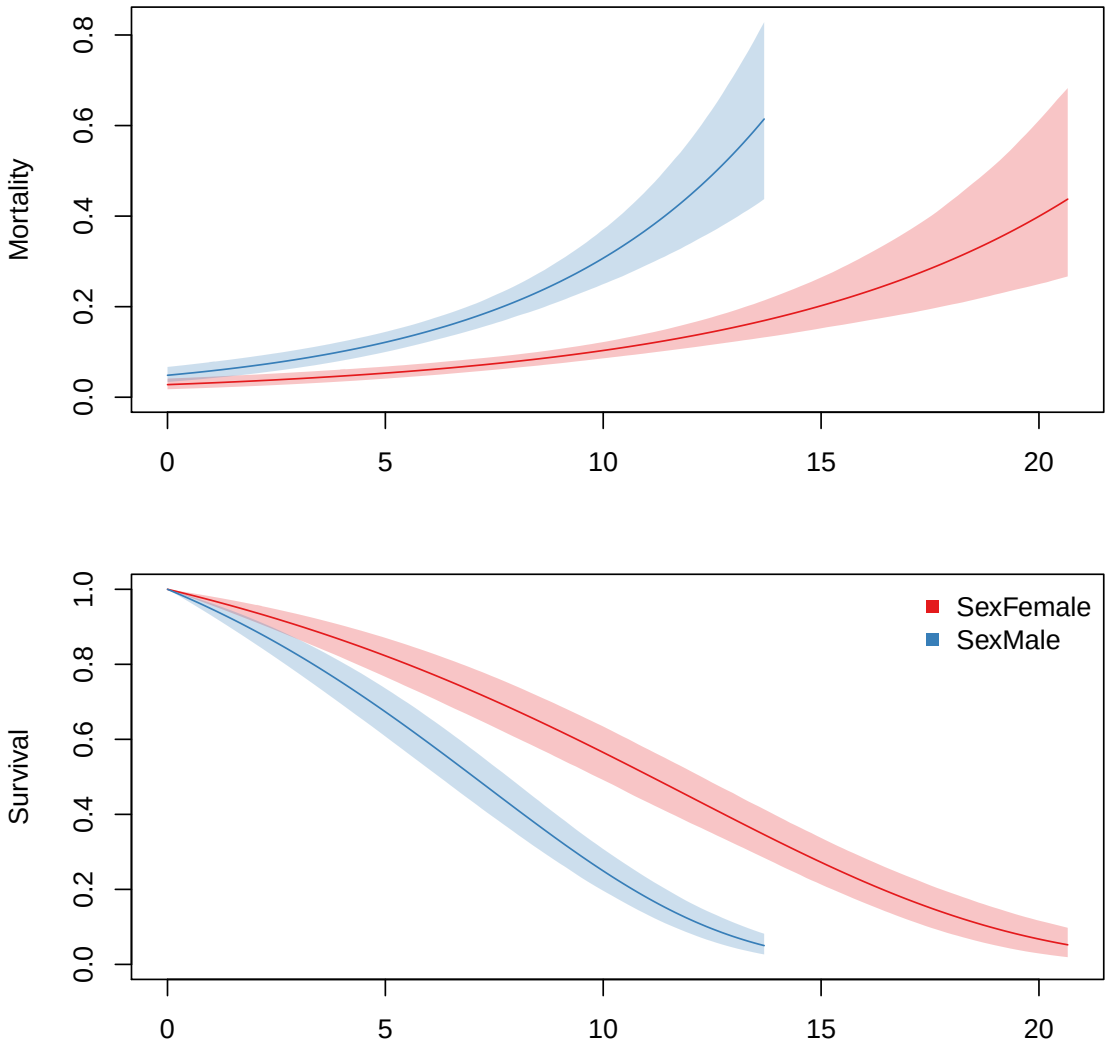

Figure 5: Survival and mortality trajectories from a simple Gompertz analysis with sex covariate.

BaSTA includes four basic mortality functions : a) exponential (Cox & Oakes 1984); b) Gompertz (Gompertz 1825, Pletcher 1999); c) Weibull (Pinder III *et al.* 1978); and d) logistic (Vaupel *et al.* 1979, Pletcher 1999) (Table 3).

Table 3: Basic mortality and survival probability functions 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	$e^{b_0+b_1x}$	$\exp\left[\frac{e^{b_0}}{b_1}(1-e^{b_1x})\right]$	$-\infty < b_0, b_1 < \infty$
Weibull	$b_0\ b_1\ (b_1x)^{b_0-1}$	$\exp\left[-(b_1x)^{b_0}\right]$	$b_0, b_1 > 0$
Logistic	$\frac{e^{b_0+b_1x}}{1+b_2\frac{e^{b_0}}{b_1}(e^{b_1x}-1)}$	$\left(1+b_2\frac{e^{b_0}}{b_1}\left(e^{b_1x}-1\right)\right)^{-1/b_2}$	$b_0, b_1, b_2 > 0$

Additional terms can be added to explore more complex shapes in the mortality function (see Fig. 2). For example, a Makeham type of structure (Pletcher 1999) consists of adding a constant to the mortality function such that

$$\mu_0(x|\mathbf{b}, c) = c + \mu_b(x|\mathbf{b}), \tag{2a}$$

$$S_0(x|\mathbf{b}, c) = e^{-cx} S_b(x|\mathbf{b}), \tag{2b}$$

with $c \leq 0$. A family of bathtub shapes can also be explored (e.g. Siler 1979). In BaSTA, these are constructed by adding a declining Gompertz function and a constant to the basic mortality, which yields:

$$\mu_0(x|\mathbf{b}, \mathbf{a}, c) = e^{a_0-a_1x} + c + \mu_b(x|\mathbf{b}), \tag{3a}$$

$$S_0(x|\mathbf{b}, \mathbf{a}, c) = \exp\left[\frac{e^{a_0}}{a_1}(e^{-a_1x}-1) - cx\right] S_b(x|\mathbf{b}), \tag{3b}$$

with $-\infty < a_0 < \infty$, $a_1 > 0$, and $c \leq 0$.

3.2 General properties of the mortality models

Each of the mortality functions evaluated in BaSTA can describe different trends in age-specific mortality (see Fig. 2). For example, the exponential model (**EX**), assumes that mortality is constant with age, which translates into an exponential decay of the survival probability in equation 1b.

The Gompertz model depicts an exponential change in mortality. If the Gompertz rate parameter (b_1 in Table 3 for Gompertz) is larger than 0, then mortality increases exponentially, accelerating also exponentially. If the rate parameter is less than 0, then mortality decreases with age, decelerating exponentially and if it is equal to 0, the model becomes a simple exponential function (Fig. 6):

$$\lim_{x\to\infty} \mu(x|b_0, b_1) = \begin{cases} \infty & \text{if } b_1 > 0 \\ b_0 & \text{if } b_1 = 0 \\ 0 & \text{if } b_1 < 0. \end{cases} \tag{4}$$

This function has been used to model age-patterns of mortality in a range of species including primates (Bronikowski *et al.* 2011), as well as in wild ungulate populations (Gaillard *et al.* 2004). Furthermore, it can

be used to test hypotheses on declining mortality with age, also called "negative senescence" (Vaupel *et al.* 2004). A risk of using the declining Gompertz model is that it can predict immortal individuals (which are not biologically reasonable) because, as equation 4 shows, mortality will tend towards 0. This problem can be solved by using a 'Makeham' structure (with the **shape** argument), which will make the model tend to the c parameter, rather than 0 at advanced ages (equation 2).

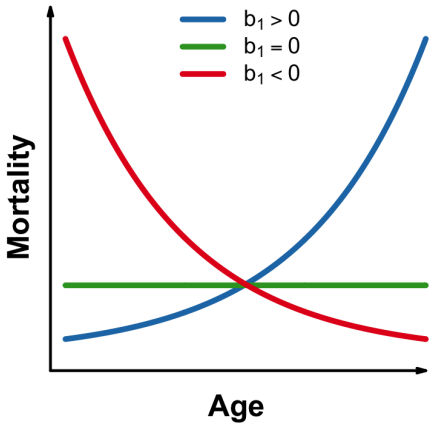

Figure 6: Mortality trajectories using a Gompertz function with varying b_1 parameters.

The Weibull model is a power function which, depending on the values of the shape parameter (b_0 in Table 3 for Weibull), can show an accelerating increase, a decelerating increase, a decrease, or constant mortality (Fig. 7):

$$\lim_{x\to\infty} \mu(x|b_0, b_1) = \begin{cases} \infty & \text{if } b_0 > 2 \\ b_0 b_1^{b_0} & \text{if } 1 < b_0 < 2 \\ b_1 & \text{if } b_0 = 1 \\ 0 & \text{if } 0 < b_0 < 1. \end{cases} \tag{5}$$

This model has been used on wild and captive bird populations (Ricklefs & Scheuerlein 2001) and can also be used to investigate or describe declining mortality. Again, we strongly recommend using a Makeham term to avoid the problem of potentially immortal individuals.

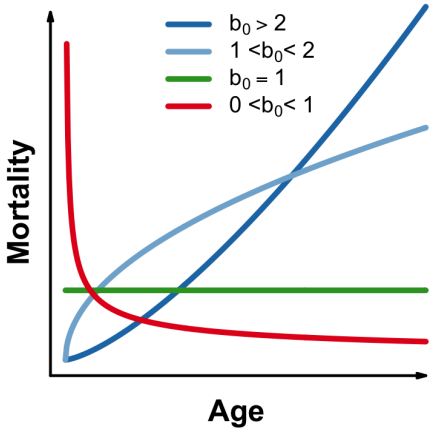

Figure 7: Mortality trajectories using a Weibull function with varying b_0 parameters.

The logistic model implemented in BaSTA depicts an initially exponential increase in mortality that

decelerates after an inflection age until it reaches a plateau. The b_2 parameter describes the degree of deceleration in mortality with age. When this parameter is equal to 0, the model becomes a simple Gompertz model (Fig. 8).

Logistic models (which are also known as Perks models; Perks (1932)) are characterized a slackening of mortality with advancing age, such that there is an asymptote or plateau at older ages (Fig. 8). The models are usually formulated as an extension of the Gompertz-Makeham model but with an extra term that governs the rate of decline and the level of the asymptote. The slackening of mortality can be interpreted as a real phenomenon (i.e. an improvement in the individuals with age; Pletcher 1999), or as an artifact caused by heterogeneity among individuals in the population (Vaupel *et al.* 1979). Beard (1959) was the first to note that Perks' model had a logistic form (Perks (1932) does not describe them as such), and that it could arise from heterogeneity among individuals in the population. He showed that if the individuals in the population were subject to Makeham mortality, but with the initial mortality parameter varying according to a gamma distribution, then the population-level average would be logistic in form. This result, a "Gamma-Makeham" model, was later independently discovered and developed by Vaupel *et al.* (1979) in a model he described as a "frailty model".

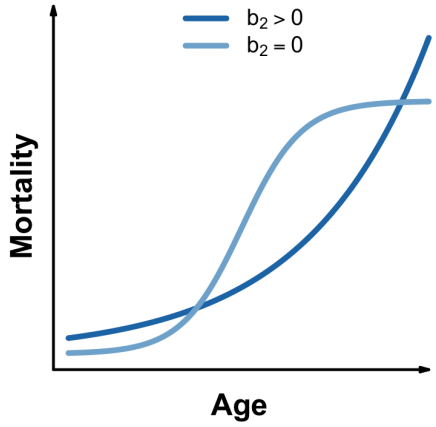

Figure 8: Mortality trajectories using a logistic function with varying b_2 parameters.

3.3 Broken stick model for survival after a minimum age

As we mentioned above, BaSTA allows users to condition the analysis to survival after a minimum age, x_m. In this case, the model uses a 'broken stick' approach whereby early mortality (or fate) is assumed to be exponential, such that $\mu_j(x|\lambda) = \lambda$, with pdf $f_j(x|\lambda) = \lambda e^{-\lambda x}$ and survival $S_j(x|\lambda) = e^{-\lambda x}$. Thus, the final pdf of ages at death (the basis for the likelihood) is calculated as:

$$f(x|\theta, \lambda) = \begin{cases} f_j(x|\lambda) & \text{if } x \leq x_m \\ S_j(x_m|\lambda) f_a(x - x_m|\theta) & \text{if } x > x_m, \end{cases} \tag{6}$$

where $f_a(x - x_m|\theta)$ is defined as in equations 1 to 3. It is important to stress that estimates of this parameter λ reflect only the rate of return of juveniles to the study site. This parameter should be used for inference on early mortality only when there is a high probability that non-returning juveniles are dead.

3.4 Covariates in mortality models

Proportional hazards are constructed as:

$$\mu(x|\boldsymbol{\theta},\boldsymbol{\eta},\mathbf{z}) = \mu_0(x|\boldsymbol{\theta})\exp\left[\boldsymbol{\eta}^T\mathbf{z}\right], \tag{7a}$$

$$S(x|\boldsymbol{\theta},\boldsymbol{\eta},\mathbf{z}) = S_0(x|\boldsymbol{\theta})^{\exp[\boldsymbol{\eta}^T\mathbf{z}]}, \tag{7b}$$

where $\mathbf{z}$ is a vector of covariates, $\boldsymbol{\eta}^T$ is a transposed vector of proportional hazards parameters and $\boldsymbol{\theta}$ is a vector of survival parameters as defined in table 3 and equations 2 and 3.

Below is an example of how to incorporate covariates as a linear function of survival parameters with a Gompertz mortality:

$$\mu(x|\boldsymbol{\theta}) = \exp\left[\overbrace{\left(\boldsymbol{\alpha}^T\mathbf{z}\right)}^{b_0} + \overbrace{\left(\boldsymbol{\eta}\boldsymbol{a}^T\mathbf{z}\right)}^{b_1} x\right], \tag{8}$$

where $\boldsymbol{\alpha}^T$ and $\boldsymbol{\eta}\boldsymbol{a}^T$ are two transposed vectors of linear coefficients that link the covariates $\mathbf{z}$ with the survival parameters b_0 and b_1 (for an extended example see Appendix S1 in Colchero & Clark 2012) such that $\boldsymbol{\theta} = \boldsymbol{\alpha} \cup \boldsymbol{\eta}\boldsymbol{a}$.

3.5 MCMC convergence diagnostic

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 ran 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.

3.6 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 realize 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).

3.7 Parameter comparison for categorical covariates

BaSTA also includes a diagnostic based on Kullback-Leibler 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 calculated as in equation 8. 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:

$$K(P_f, P_m) = \int_0^\infty P_f \log\left(\frac{P_f}{P_m}\right) d\alpha. \tag{9}$$

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 9, 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 $q(k)$ be 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:

$$q(k) = \frac{(1 + (1 - e^{-2k})^{\frac{1}{2}})}{2}. \tag{10}$$

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.

References

Beard, R. (1959) *Note on some mathematical mortality models*, pp. 302–311. Ciba Foundation Colloquium on Ageing.

Bronikowski, A.M., Altmann, J., Brockman, D.K., Cords, M., Fedigan, L.M., Pusey, A., Stoinski, T. & *et al.* (2011) Aging in the Natural World: Comparative Data Reveal Similar Mortality Patterns Across Primates. *Science*, **331**, 1325–1328.

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. (2012) Bayesian inference on age-specific survival for censored and truncated data. *Journal of Animal Ecology*, **81**, 139–149.

Colchero, F., Jones, O.R. & Rebke, M. (2012) BaSTA: an R package for Bayesian estimation of age-specific mortality from incomplete mark-recapture/recovery data with covariates. Method in Ecology and Evolution, **3**, 466–470.

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

Gaillard, J., Viallefont, A., Loison, A. & Festa-Bianchet, M. (2004) Assessing senescence patterns in populations of large mammals. *Animal Biodiversity and Conservation*, **27**, 47–58.
URL `http://www.raco.cat/index.php/ABC/article/viewArticle/56918/0`

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.

Perks, W. (1932) On Some Experiments on the Graduation of Mortality Statistics. *Journal of the Institute of Actuaries*, **63**, 12–40.

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/`

Ricklefs, R. & Scheuerlein, A. (2001) Comparison of aging-related mortality among birds and mammals. *Experimental Gerontology*, **36**, 845–857.

Roberts, G.O. & Rosenthal, J.S. (2009) Examples of Adaptive MCMC. *Journal of Computational and Graphical Statistics*, **18**, 349–367.

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.

Vaupel, J., Manton, K. & Stallard, E. (1979) The impact of heterogeneity in individual frailty on the dynamics of mortality. *Demography*, **16**, 439–454.
URL `http://www.jstor.org/stable/2061224`

Vaupel, J.W., Baudisch, A., Dölling, M., Roach, D.A. & Gampe, J. (2004) The case for negative senescence. *Theoretical Population Biology*, **65**, 339?351.

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