survHE: Survival analysis for health economic evaluation and cost-effectiveness modelling

Gianluca Baio
University College London

Abstract

The abstract of the article.

Keywords: Survival analysis, health economic evaluation, Probabilistic Sensitivity Analysis, R.

1. Introduction

Broadly speaking, the objective of publicly funded health care systems (such as the UK's) is to maximise health gains across the general population, given finite monetary resources and a limited budget. Bodies such as the National Institute for Health and Care Excellence (NICE) provide guidance on decision-making on the basis of health economic evaluation. This covers a suite of analytical approaches for combining costs and consequences of intervention(s) compared to a control or status quo, the purpose of which is to aid decision making associated with resource allocation. To this aim, much of the recent research has been oriented towards building the health economic evaluation on sound and advanced statistical decision-theoretic foundations, arguably making it a branch of applied statistics (Willan and Briggs 2006; Briggs et al. 2006).

Interventions that impact upon survival form a high proportion of the treatments appraised by NICE (Latimer 2011). Interestingly, in order to quantify accurately the economic benefits of a new intervention, it is necessary to estimate the mean survival time (rather than usual summaries, such as the median time). Thus, it is necessary to extrapolate the observed survival curve (which often only covers a limited time frame and is subject to high degree of censoring) to a longer time horizon. Consequently, a parametric approach to the survival analysis is usually followed and it is recommended by NICE's guidelines (Latimer 2011).

A Bayesian approach is particularly helpful in economic evaluation because it allows to fully characterise uncertainty in the model parameters. This in turn is a fundamental component of the cost-effectiveness analysis, because it is crucial to assess the impact of this uncertainty on the decision-making process (Briggs et al. 2006; Baio 2012), a process termed Probabilistic Sensitivity Analysis (PSA). However, parametric survival analysis can be challenging from a Bayesian point of view, in terms of computation.

Perhaps for this reason, frequentist methods to perform survival analysis with an emphasis in economic evaluations represent the industry standard. In particular, survival analysis is often embedded in health economic evaluations using a multi-step/multi-software approach: first estimates from a survival model are considered. These are often obtained by published clini-

cal studies presenting point and interval estimates for the model parameters. Modellers then usually produce a set of simulations for the model parameters using Monte Carlo (MC) procedures. Finally, these simulations are used to produce a large number of survival curves, which are fed to the economic model (for example to determine the benefits of a given intervention). The whole process is typically performed in `Excel`.

This is less than ideal. Firstly, in the MC simulations, potential correlation among the model parameters is only approximated and not fully accounted for potentially introducing bias in the estimate for the survival curves and thus in the outcome of the decision-making process. A full Bayesian approach, e.g. based on Markov Chain Monte Carlo (MCMC) would eliminate this issue because the inference would be produced directly on the full joint distribution of the model parameters. In addition, the clinical evidence used to inform the survival analysis for health economic evaluation can be limited, thus emphasising the problem of uncertainty in the extrapolation of the survival curves. The inclusion of prior information (*e.g.* to encode the assumption that cancer patients' survival should be well below that of the healthy population, or from evidence on drugs with similar therapeutic mechanisms), which is instrumental to a Bayesian analysis, would again improve the model performance. Non-standard models, such as the Poly-Weibull (Demiris et al. 2015), may also be effectively implemented and used within a Bayesian approach.

The limitations of spreadsheets such as `Excel` in terms of statistical modelling (and particularly in survival analysis) are increasingly often recognised in the health economics literature (Williams et al. 2016; Baio and Heath 2016). With this in mind, the objective of this work is to develop a suite of functions and tools for the freely available statistical software R, specifically designed for the needs of modellers using survival analysis results to build extensive models for health economic evaluation.

2. The R package survHE

survHE is an R package specifically designed to aid in the process of survival analysis for health economic evaluation and cost-effectiveness analysis. In fact, **survHE** can be actually considered as a wrapper for some other R packages; the first one, **flexsurv** (Jackson 2016), is in turn a general-purpose tool for performing several types of survival analysis using maximum likelihood estimates (MLEs). The second one, **rstan** (Carpenter et al. 2015) is a relatively new R package that can be used to perform Bayesian analysis using Hamiltonian Monte Carlo (HMC). This is a form of Markov Chain Monte Carlo algorithm, which can be used to produce samples from a joint posterior distribution of a set of model parameters and unobserved quantities. Finally, the third one, **INLA** (Martins et al. 2013) can be used to perform fast Bayesian computations (on a limited set of survival models) using Integrated Nested Laplace Approximation.

In a sense, thus, **survHE** is a very simple package that specialises functions from other relevant packages and builds some other specific commands to simplify and standardise the process of using survival data in health economic evaluation. There are two ways of installing **survHE**. A "stable" version is packaged and binary files are available for Windows and as source (at present, this is version `1.0.4`). To install the stable version on a `Windows` machine, run the following commands

```
> install.packages("survHE",
```

```
+                   repos=c("http://www.statistica.it/gianluca/R",
+                           "https://cran.rstudio.org",
+                           "https://www.math.ntnu.no/inla/R/stable"),
+                   dependencies=TRUE
+ )
```

Note that it is necessary to specify a vector of repositories — the first one hosts **survHE**, while the second one should be an official CRAN mirror (*e.g.* see https://cran.r-project.org/index.html) to ensure that the R command `install.packages()` can also install the "dependencies" (*e.g.* other packages that are required for **survHE** to work). The third one is used to install the package **INLA**. This process can be quite lengthy, if you many of the relevant packages are not already installed.

The second way involves using the "development" version of **survHE**— this will usually be updated more frequently and may be continuously tested. On Windows machines, the following code will work.

```
> pkgs <- c("flexsurv","Rcpp","rms","xlsx","rstan","INLA","Rtools","devtools")
> repos <- c("https://cran.rstudio.com", "https://www.math.ntnu.no/inla/R/stable")
> install.packages(pkgs,repos=repos,dependencies = "Depends")
```

before installing the package using the package `devtools`, by typing the following command.

```
> devtools::install_github("giabaio/survHE")
```

As a first example, suppose that the user has a suitable dataset, perhaps obtained from a trial, in which data are recorded for the time at which observations are made, a censoring indicator taking value 1 if an event (*e.g.* progression to a cancerous state, or death) has actually occurred at that time and 0 if the individual has been censored (*e.g.* we have not observed any event at the end of follow up) and an arm indicator, specifying whether the individual to whom the data refer belongs in the control or the active treatment arm of the trial. Of course, other variables may be observed, *e.g.* relevant covariates, such as sex, age, co-morbidity.

For the moment we consider the simple case in which the data are available in the R workspace as a data frame (say, `data`) that can be partially visualised using the following command:

```
rbind(head(data),tail(data))
```

to show the first 6 and the last 6 rows:

```
    ID_patient  time censored arm
1            1  0.03        0   0
2            2  0.03        0   0
3            3  0.92        0   0
4            4  1.48        0   0
5            5  1.64        0   0
6            6  1.64        0   0
362        362 13.97        1   1
363        363 14.56        1   1
364        364 14.56        1   1
365        365 14.85        1   1
366        366 16.07        1   1
367        367 18.16        1   1
```

In this case, the dataset consists of 367 individuals in total, grouped in two arms (here `arm` = 0 indicates the controls and `arm` = 1 indicates the active treatment).

2.1. Modelling survival data: to be or not to be (Bayesian)?...

By relying on either **flexsurv**, **INLA** or **rstan**, **survHE** can fit models under the frequentist or Bayesian framework.

In general terms, a Bayesian approach has several advantages: for example, health economic evaluations are typically based on complex models, often made by several (correlated) modules, which may be informed by different and diverse sources of evidence. Thus, a Bayesian approach can be beneficial to propagate the underlying uncertainty in all the model parameters in a principled way. This is also particularly relevant in terms of what is often termed "probabilistic sensitivity analysis" (PSA) in health economics (Baio and Dawid 2011; Baio 2012). That is the practice of assessing the impact of parameters uncertainty on the decision-making process and is usually based on a simulation approach to characterise the underlying uncertainty in the model parameters — a fundamentally Bayesian operation.

On the other hand, in addition to the need of specifying suitable prior distributions that are consistent with the information available for the case at hand, Bayesian models for survival analysis fitted using MCMC can be computationally intensive and sometimes have problems with convergence; perhaps for this reason, often practitioners use MLE-based routines to obtain relevant estimates from the survival component of the wider economic model. These are usually simple to obtain. In order to deal with PSA, **flexsurv** uses bootstrap (based on multivariate Normal distributions), which may be a good approximation of the underlying full joint posterior distribution of the survival parameters.

A good compromise between frequentist and fully Bayesian models is provided by **INLA**, which is effectively an alternative method of performing Bayesian inference. By using a specific (albeit rather general), clever model specification and an approximation algorithm, **INLA** typically requires a computational time that is very close to that of MLE-based routines, while also estimating an approximation to the full joint posterior distribution for the model parameters. However, in general terms, it is a bit more complex to embed **INLA** within a more complex model; in addition, currently, **INLA** can only fit a limited number of survival models.

For these reasons, we chose to design **survHE** around these three approaches: ideally, we would build the whole economic model under a Bayesian framework and take full advantage of the flexibility provided by MCMC estimation — this would be naturally obtained by using **rstan**. Note that we choose **rstan** over other software such as `OpenBUGS` or `JAGS`; the reason for this is that HMC often proves a superior mode of inference compared to Gibbs Sampling (the MCMC algorithm upon which `OpenBUGS` and `JAGS` are based). In particular, for the specific set of models that we consider here, HMC proves faster and more reliable in terms of convergence than Gibbs Sampling. In addition, **rstan** models can be "pre-compiled" and thus the computational time required is totally devoted to sampling from the relevant posterior distributions, which makes a full Bayesian approach more competitive (in comparison to MLE-based methods).

However, despite these very useful features of **rstan**, we acknoledge that at times Bayesian survival models can still be challenging from the point of view of computation (especially with large dataset and for particularly complex model structures). In these cases, **INLA** is very helpful, for the range of distributions it supports; similarly, particularly in specific settings,

some practitioners may want to use a standard approach to statistical inference and MLEs.

The combined use of **survHE** and its "dependencies" (*i.e.* the other packages on which it depends, in R parlance) allows all of these options in a unified framework. More importantly, irrespective of the way in which the inference on the survival model is performed, **survHE** has a set of built-in functions that can be used to produce a standardised post-processing of the results, for their inclusion in the economic model and PSA.

2.2. Modelling framework

The general modelling framework considered in **survHE** can be described as follows. The observed data are at least made by the pair (t_i, d_i), for $i = 1, \ldots, n$, where $t_i > 0$ is the observed time at which the event under study occurs and d_i (for "dummy" variable) is an event indicator, taking value 1 if the event actually happens (in which case t_i is indeed observed), or 0 when the $i-$th individual is "censored". In this latter case, we do not know whether the event actually occurs — it may in the future, but we just do not have this information. Consequently, when $d_i = 0$, then the observed t_i does not represent the true "survival time". Notice here that we consider for the sake of simplicity "right censoring", which is the most common for applications in health economics.

The observed data are modelled using a suitable probability distribution $t_i \sim f(t_i \mid \boldsymbol{\theta})$, as a function of a vector of relevant parameters $\boldsymbol{\theta}$. This can be linked to the *survival function*

$$S(t) = 1 - F(t) = 1 - \int_0^t f(u \mid \boldsymbol{\theta}) du,$$

indicating the probability of an individual surviving up to time t, as well as to the *hazard function*

$$h(t) = \frac{f(t)}{S(t)},$$

which quantifies the instantaneous risk of experiencing the event. In the presence of censoring, the resulting log-likelihood function is modified to account for the possibility of partially observed data (in correspondence with censoring) and is expressed as

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[d_i \log h(t_i) + \log S(t_i) \right]. \tag{1}$$

This basically models the risk of experiencing the event at any time point t, conditionally on the fact that the $i-$th unit has in fact survived up to that time point (if they have not, then the probability of experiencing the event again is 0).

When formulating a parametric survival model, we need to specify the form of the probability distribution assumed to describe the underlying uncertainty in the observed times. As mentioned above, it is good practice to test a set of (more or less) plausible parametric models for the survival data. This is the procedure recommended by NICE guidelines (Latimer 2011).

In general terms, we can specify the vector of relevant parameters as $\boldsymbol{\theta} = (\mu(\boldsymbol{x}), \alpha(\boldsymbol{x}))$. In this notation, we consider: a vector of potential covariates $\boldsymbol{x}$ (*e.g.* age, sex, trial arm, etc.); a *location* parameter $\mu(\boldsymbol{x})$, which indicates the mean or the scale of the probability distribution; and a (set of) ancillary parameter(s) $\alpha(\boldsymbol{x})$, which describes the shape or variance of the distribution. While it is possible for both μ and α to explicitly depend on the covariates $\boldsymbol{x}$, usually the formulation is simplified to assume that these only affect directly the location parameter.

In addition, since $t > 0$, we typically use a generalised linear model in the form

$$g(\mu_i) = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ij} [+ \ldots] \tag{2}$$

to model the location parameter. The function $g(\cdot)$ is typically the logarithm; notice that here we slightly abuse the notation and omit the dependence of μ_i on $\boldsymbol{x}$. Generally speaking, (2) can be extended to include additional terms — for instance, we may want to include random effects to account for repeated measurements or clustering. We indicate this possibility using the $[+\ldots]$ notation and highlight the fact that this is rather straightforward in a Bayesian context (*i.e.* both for **INLA** and **rstan**).

The objective of the statistical analysis is the estimation of the parameters $\boldsymbol{\theta}$, which can then be used to obtain estimates for all the relevant quantities (*e.g.* the survival function), which are then in turn used in the economic modelling, for example to estimate the transition probabilities in a Markov model.

In a frequentist setting, the estimation procedure concerns some relevant statistics, *i.e.* functions of the observed data and is performed via maximum likelihood estimation. Conversely, in a full Bayesian setting, the parameters are directly modelled using a prior probability distribution, which is updated by the observed data into a posterior. It is this posterior distribution that is the object of the inferential process. Thus, when using a Bayesian framework, the model needs to be completed by specifying suitable prior distributions for the parameters $\boldsymbol{\theta}$.

Assuming that the location parameter is specified using a linear predictor form, on the scale determined by the function $g(\cdot)$ and as a function of J covariates, we can model $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_J) \overset{iid}{\sim} \text{Normal}(\mu_\beta, \sigma_\beta)$. Note that **survHE** expands any categorical covariates to a set of dummy variables: so if a covariate has four categories, in line with R notation, **survHE** considers three binary indicators. Thus the profile `(0,0,0)` indicates the first (reference) category, while the profiles `(1,0,0)`, `(0,1,0)` and `(0,0,1)` indicate the second, third and fourth category, respectively.

In both its Bayesian versions, **survHE** assumes by default $\mu_\beta = 0$ and $\sigma_\beta = 5$ for the models in which the linear predictor is defined on the log scale and $\sigma_\beta = 100$ for those in which the linear predictor is defined on the natural scale and thus $g(\cdot)$ is the identity function. This amounts to specifying a "minimally informative" prior on the regression coefficients that determine the location parameter — in other words, we are not including strong prior information in this aspect of our model. The observed data (and the censoring structure) will be mainly driving the update to the posterior distribution. When genuine prior knowledge is present, *e.g.* about the likely size of a treatment effect, it is possible to modify these priors to encode the information in the model formulation.

As for the ancillary parameter, the choice of prior depends on the specification of the probability distribution selected to model the observed data. Table 1 shows a summary of the models directly implemented in **survHE**. In each, by default we specify minimally informative priors on the relevant parameters; for example, in the Weibull model, we define $\alpha \sim \text{Gamma}(a, b)$, for given values of a, b.

All the models presented in Table 1 are available using MLE and HMC as inferential engines. On the other hand, **INLA** currently only handles Exponential, Weibull, log-Normal and log-Logistic models. The names presented in the rightmost column can be used when calling

Data model	Location parameter	Ancillary parameter*	**survHE** name
$t_i \sim$ Exponential(μ_i)	Rate: $\mu_i = \exp\left(\beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}\right)$	–	`exp`, `exponential`
$t_i \sim$ Weibull(μ_i, α)	Scale: $\mu_i = \exp\left(\beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}\right)$	Shape: $\alpha \sim$ Gamma$(0.1, 0.1)$	`weibull`, `weibullPH`
$t_i \sim$ logNormal(μ_i, α)	log-mean: $\mu_i = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}$	log-sd: $\alpha \sim$ Uniform$(0, 5)$	`lnorm`, `lognormal`
$t_i \sim$ logLogistic(μ_i, α)	Rate: $\mu_i = \exp\left(\beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}\right)$	Shape: $\alpha \sim$ Gamma$(0.1, 0.1)$	`llogis`, `loglogistic`
$t_i \sim$ Gamma(μ_i, α)	Rate: $\mu_i = \exp\left(\beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}\right)$	Shape: $\alpha \sim$ Gamma$(0.1, 0.1)$	`gamma`
$t_i \sim$ Gompertz(μ_i, α)	Rate: $\mu_i = \exp\left(\beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}\right)$	Shape: $\alpha \sim$ Normal$(0, 5)$	`gompertz`
$t_i \sim$ Gen Gamma$(\mu_i, \boldsymbol{\alpha})$	Location: $\mu_i = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}$	$\boldsymbol{\alpha} = (\sigma, q)$ Scale: $\sigma \sim$ Gamma$(0.1, 0.1)$ Shape: $q \sim$ Normal$(0, 100)$	`gengamma`, `gengamma.orig`
$t_i \sim$ Gen F$(\mu_i, \boldsymbol{\alpha})$	Location: $\mu_i = \beta_0 + \sum_{j=1}^{J} \beta_j x_{ij}$	$\boldsymbol{\alpha} = (\sigma, q, p)$ Scale: $\sigma \sim$ Gamma$(0.1, 0.1)$ Shape(1): $\log(p) \sim$ Normal$(0, 0.5)$ Shape(2): $q \sim$ Normal$(0, 2.5)$	`genf`, `genf.orig`

* The distributions presented for the ancillary parameters are the default used by **survHE**

Table 1: A list of distributions supported by **survHE**

survHE. We mostly follow the notation of **flexsurv**, but also allow for some specific differences in the **INLA** notation: for example, the log-Logistic distribution can be referred to using **flexsurv** (`llogis`), or **INLA** notation (`loglogistic`). **survHE** will internally map the different strings of text and select the correct routine.

Notice also that, in line with **flexsurv**, **survHE** allows the two versions of the Weibull model, *i.e.* using an Accelerated Time Failure (AFT, `weibull`) or a Proportional Hazard (PH, `weibullPH`) parameterisation. When the interest is in estimating the effect of some covariates on the survival time, these two versions yield of course different estimates. However, in the case of health economic evaluation, the interest is really in producing an estimate of the distribution of the survival curves. **survHE** internally maps the estimated coefficients to the correct transformation so as to estimate $S(t)$ correctly, irrespective of the original parametrisation. For example, when the AFT parameterisation is used, then $S(t) = \exp\left(-\left(\frac{t}{\mu_i}\right)^{\alpha}\right)$, while for the PH parameterisation $S(t) = \exp\left(-\mu_i t^{\alpha}\right)$.

The "original" parameterisation of the Generalised F and Generalised Gamma distributions (respectively `genf.orig` and `gengamma.orig`) are currently available only when the estimation is performed using MLE in **flexsurv** (this is mostly for backward compatibility).

3. MLE via flexsurv

survHE allows the user to define in R a vector of model names (in the format that **flexsurv** or **INLA** can recognise). We could for instance decide that we want to consider the Exponential, Weibull, Gamma, log-Normal, log-Logistic and Generalised Gamma models for our analysis.

We can do this in R by using the following commands.

```
> mods = c("exp","weibull","gamma","lnorm","llogis","gengamma")
```

This syntax defines a vector of model names to be used by **flexsurv** in the fitting process. These *must* adhere with **survHE** convention, as specified above.

At this point, we are almost ready to actually perform the survival analysis using the 6 models specified above; before we can do this, however, we need to specify the model "formula", for example as the following.

```
> formula = Surv(time, censored) ~ as.factor(arm)
```

This creates an object instructing R to analyse the data using a survival model in which the only covariate is the treatment arm, interpreted as an R "factor" (*i.e.* a categorical variable).

The **survHE** function `fit.models` can be used to actually perform this analysis in batch, *e.g.* by typing the command

```
> m1 = fit.models(formula=formula,data=data,distr=mods)
```

The function `fit.models` takes as mandatory inputs the analysis formula, the name of the dataset to be used and the type of distribution(s) to be fitted. Just like in this case, this may be a vector, in which case `fit.models` will store all the different analyses in the resulting object.

Executing the command above creates an object `m1` in the class `survHE`, in which the results of the survival analyses are stored for each parametric model considered. The usual R command

```
> names(m1)
```

returns the names of the several elements in the list.

```
[1] "models"        "model.fitting" "method"        "misc"
```

The object `models` is itself a list, in this case containing 6 elements (one for each of the parametric models fitted). For example, the first one can be accessed using the standard R notation `m1$model[[1]]` (*i.e.* using "double square brackets") and can be inspected typing the command

```
names(m1$models[[1]])
```

```
 [1]  "call"           "dlist"          "aux"            "ncovs"
 [5]  "ncoveffs"       "mx"             "basepars"       "covpars"
 [9]  "AIC"            "data"           "datameans"      "N"
[13] "events"         "trisk"          "concat.formula" "all.formulae"
[17] "dfns"           "res"            "res.t"          "cov"
[21] "coefficients"   "npars"          "fixedpars"      "optpars"
[25] "loglik"         "logliki"        "cl"             "opt"
```

The quantities included in the model objects are the standard output from **flexsurv**. Typically, the user does not need to access or manipulate them directly (that is the point of **survHE**!); in fact, other **survHE** functions will use these to produce plots or further analyses. Users familiar with R programming can however access them to post-process their results and customise even further the output provided by **survHE**.

The other elements of the object `m1` are:

- `model.fitting`: a list storing some suitable statistics that can be used to assess, well, model fitting. These are the Akaike, Bayesian and Deviance Information Criteria (AIC, BIC and DIC, respectively). The former two can be estimated using both the Bayesian and frequentist approach, while the latter is specific to Bayesian modelling. Thus, in this case, the R call

  ```
  > m1$model.fitting
  ```

  will return the following results

  ```
  $aic
  [1] 1274.576 1203.130 1203.504 1214.984 1208.494 1204.785

  $bic
  [1] 1282.387 1214.846 1215.220 1226.700 1220.211 1220.406

  $dic
  [1] NA NA NA NA NA NA
  ```

  — note that because we are storing the results obtained from fitting 6 models in the same object, the elements `$aic`, `$bic` and `$dic` are vectors. In general, the model associated with the lowest information criterion tends to be associated with a better fit, as we discuss in §6.4.

- `method`: a string describing the method used to fit the model(s). In this case the code

  ```
  > m1$method
  ```

  returns the output

  ```
  "mle"
  ```

- `misc`: a list containing some miscellanea — these are mainly used internally by the other functions in **survHE** to do plots and tables or other calculations. Specifically, the elements of this objects are

  - `time2run`: the time used to run the model(s) (in seconds);
  - `formula`: the R object containing the formula used to define the model(s);
  - `km`: the Kaplan-Meier estimate produced automatically by **survHE** (using the function `npsurv` from the R package **rms**);
  - `data`: the data-frame containing the original data used to fit the model(s).

4. Bayesian analysis via INLA

4.1. Integrated Nested Laplace Approximation

Integrated Nested Laplace Approximation (INLA; Rue et al. 2009) can be used to perform direct numerical calculation of posterior densities in a wide sub-class of Bayesian hierarchical models (called Latent Gaussian Models, LGMs), avoiding time-consuming Markov chain Monte Carlo simulations. INLA implementation covers models of the form

$$\begin{aligned} y_i \mid \boldsymbol{\theta}, \boldsymbol{\phi} &\sim p(y_i \mid \boldsymbol{\theta}, \boldsymbol{\phi}) \\ \boldsymbol{\theta} \mid \boldsymbol{\phi} &\sim \text{Normal}\left(0, \boldsymbol{Q}^{-1}(\boldsymbol{\phi})\right) \\ \boldsymbol{\phi} &\sim p(\boldsymbol{\phi}), \end{aligned}$$

where y_i is the observed variable, $\boldsymbol{\theta}$ is a set of parameters (which may and often has a very large dimension) and $\boldsymbol{\phi}$ is a set of hyperparameters. The main restrictions of the INLA formulation are the fact that the number of hyperparameters needs to be limited (for computational convenience) and the form of the prior imposed on the parameter. This is a multivariate Normal distribution where the precision matrix $\boldsymbol{Q}^{-1}(\boldsymbol{\phi})$ depends on the hyperparameters and exploits conditional independences across the parameters. This general structure is called a Gaussian Markov Random Field (GMRF; Rue and Held 2005).

The basic principle is to approximate the posterior density for $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ using a series of nested Normal approximations. The algorithm uses numerical optimisation to find the mode of the posterior, while the marginal posterior distributions are computed using numerical integration over the hyperparameters. INLA is a very fast method of inference and can be applied to many models that can be written in the form of LGMs — for example generalised linear models (including structured components, such as simple random effects, as well as spatial or temporal effects).

On the other hand, not all models can be easily framed within the LGM formulation. In addition, INLA's estimate are, by definition, approximations to the full joint posterior distribution of the model parameters. Thus, there is a trade-off between the computational complexity and the accuracy of the estimation. In many general cases, INLA produces a good compromise by allowing a good level of accuracy (often comparable with simulation methods such as MCMC that, if run for long enough, are guaranteed to give the "exact" value) and running time.

4.2. Using survHE to fit models with INLA

When fitting models using a Bayesian approach via **INLA**, **survHE** allows the user to select a vector of distributions; as mentioned above, currently, **INLA** and its R implementation allow four survival models: Exponential, Weibull, log-Normal and log-Logistic. The user can specify a vector `distr = c("exp","weibull","lognormal","loglogistic")`, or select only one of those models and may be run the `fit.models` command separately for each of them. If a distribution is specified that is not allowed in **INLA**, then **survHE** will fall back on the MLE specification and use **flexsurv** instead.

One important distinction is in the way in which **flexsurv** and **INLA** handle the names of the distributions to be fitted and the formula specifying the model. As for the latter, the correct notation is the following: `"exponential"`, `"weibull"`, `"lognormal"` and `"loglogistic"` —

these do not directly match the **flexsurv** notation mentioned above. As mentioned above, to avoid issues, **survHE** recodes internally the names given to the distributions. Thus, if the user specifies the additional option `method="inla"` in the call to `fit.models`, then the string `"exp"` (which would be accepted by **flexsurv**) will be recoded to `"exponential"`, which is required by **INLA**. Similarly if a distribution that is accepted by **flexsurv** is given by the user in **INLA** terminology but the `method` is either unspecified or specifically set to `"mle"` in the call to `fit.models`, then **survHE** will recode the name of the distribution in **flexsurv** terms.

With regards to the model formula, **INLA** requires that this is specified using the notation

```
> formula = inla.surv(time,event) ~ ...
```

where `time` and `event` are the variables in which the times and the event indicator are stored and `...` is a suitable form for the combination of covariates to be used. Again, **survHE** tries to simplify the modeller's life by fixing the code provided for the formula, depending on the value specified for the option `method`. So if `method="inla"` but the formula is specified using the **flexsurv** terminology, **survHE** will recode this to make it acceptable to **INLA**.

A suitable call to `fit.models` using **INLA** is the following.

```
> m2 = fit.models(formula=formula,data=dat,distr=distr.inla,method="inla")
```

where `distr.inla` is a vector of strings containing names from the four models available in **INLA**.

When `method` is set to `"inla"`, then other options become available to the call to `fit.models`, which allow the user to customise the underlying setting of the Bayesian model and inferential procedure. In particular, it is possible to add the following arguments.

- `dz`; this defines the step length for the grid search over the hyperparameters space (default = 0.1). As mentioned above, **INLA** estimates the value of the hyperparameters in the model (*e.g.* the shape of a Weibull distribution), using a grid search. The finer this grid, the more accurate (but more computationally expensive!) the resulting estimates for all the parameters, *e.g.* for both the shape and scale of the Weibull distribution.

- `diff.logdens`; defines the difference in the log-density for the hyperparameters to stop integration (default = 5). Again, this is related to how the hyperparameters are estimated in the first stage of the nested algorithm. Decreasing this difference is likely to increase the computational time, since the estimation of the hyperparameters will become more accurate.

- `control.fixed`; defines the default for the prior distributions, unless specified by the user. By default, **INLA** assumes that "fixed effects" associated with covariates are modelled using a Normal with mean 0 and variance 1000, while the overall intercept is modelled using a Normal with 0 mean and even smaller precision. **survHE** overrules this and sets the precision of the covariates in the linear predictor to $1/5^2 = 0.04$ — this is consistent with the default setting used when HMC is selected as the inferential engine.

- `control.family`; a list of options controlling the model for the observed data. If `distr` is a vector, this can be provided as a named list of options; for example:

```
> m2 = fit.models(formula=formula,data=dat,distr=distr,method="inla",
+     control.family=list(
+                     weibull=list(param=c(.1,.1)),
+                     lognormal=list(initial=2)
+     )
)
```

would instruct **INLA** to assume a Gamma$(0.1, 0.1)$ prior distribution for the shape parameter of the Weibull model and to use an initial value of 2 for the approximation routine of the log-Normal model. Notice that in this case, the names of the elements of the list need to be the same as those given in the vector `distr`.

Using **INLA** advanced controls is very powerful and allows much flexibility in fitting the models. However, some knowledge and understanding of the **INLA** syntax and philosophy is required. Guidance is provided in the **INLA** help functions as well as, for example, in Rue et al. (2009); Blangiardo et al. (2013) and Blangiardo and Cameletti (2015).

Back to the running example, we may fit the models in **INLA** using the following code.

```
> m2=fit.models(formula,data,c("exp","weibull","llogis","lnorm"),method="inla",
+    control.family=list(
+                exponential=list(),
+                weibull=list(param=c(.1,.1)),
+                loglogistic=list(),
+                lognormal=list(initial=1)
+    )
)
```

Note that it may be necessary to fiddle with the `control.family` option to successfully fit some of these models — for example, without specifying the initial value for the log-Normal model, in the case of the data given in the object `data`, **INLA** (and thus `fit.models`) would crash.

5. Bayesian analysis via HMC

5.1. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC; Radford 2011) is one of the algorithms belonging to the general class of MCMC methods. In a nutshell, HMC is based on the physical concept of Hamiltonian dynamics, which can be used to model the idealised situation of a frictionless particle sliding over a surface of varying slope. Basically, the movements of the particle depend on: *i)* the *potential energy*, a function of its current location l, which is proportional to the height of the surface at the current position; and *ii)* the *kinetic energy*, a function of its momentum m, depending on the mass of the particle. The way in which these movements happen can be described by a set of ordinary differential equations: this means that if we are able to compute the derivatives of these two functions and given a set of initial conditions specifying the starting location l_0 and momentum m_0, at time t_0, then we can predict the location and momentum of the particle at any point in time, by simulating these dynamics for a given duration.

Leaving aside all the technical difficulties, the basic intuition behind HMC is the following: the surface of interest is the *unnormalised* posterior log-density for the parameters in the model

$$\log \tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{t}) = \log p(\boldsymbol{\theta}) + \log p(\boldsymbol{t} \mid \boldsymbol{\theta}). \tag{3}$$

In general, we are not in a position of knowing the target distribution $\log \tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{t})$ exactly and in closed form[1]. Moreover, even if we were, this would only be proportional to the actual posterior density for the parameters (because the expression above is computed without rescaling by the marginal log-density for the observed data $\boldsymbol{t}$; for this reason, we use the notation $\tilde{p}$).

However, both log-densities on the right-hand side of (3) are known because they are part of the model specification. If we can compute the derivatives of $\log \tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{t})$, given initial values for the location of the parameters $\boldsymbol{\theta}$ and their momentum, we can simulate Hamiltonian dynamics. As it turns out, this is extremely efficient at exploring the (negative) posterior log-density, by proposing a move to a new position that is determined by letting the "particle" slide over the density.

This means that if the current position is far away from the portion of the parametric space where most of the probability mass lies, the potential energy is large and thus the "particle" will have higher speed when sliding over a very steep surface. More specifically, unlike simpler (and far less efficient methods) such as the Metropolis algorithm, the proposed moves will not necessarily be characterised by symmetrical distributions and will tend to be pulled towards the mode of the joint posterior distribution more quickly.

In practice, HMC can be very complex, because in addition to the specific computation of possibly complex derivatives, it requires fine tuning of several parameters. However, **rstan** provides a very clever system in which most of the adaptation is automatic. The user can still specify some of the basic inputs (and at times this is crucial to improve, or even reach convergence to the target posterior distributions), but **rstan** is a very general system to perform HMC estimation on a very wide range of models.

5.2. Using survHE to fit models with HMC

Much of the work performed by **rstan** consists in determining a set of derivatives from the model structure, that are used to apply the Hamiltonian dynamics and explore the parametric space in an efficient way. This requires a preparatory step, which **rstan** does by compiling the model in `C++` (via `R`). This step can be quite lengthy, but interestingly it is possible to pre-compile a model — if all that changes is the data (but not the structure and the distributional assumptions), then the pre-compiled model can be used directly, thus saving substantial computation time.

This is another attractive feature of **rstan**, because it means that **survHE** can pre-compile all the standard models presented in Table 1. Thus, it is possible to estimate them by using a command such as the following.

```
> m3=fit.models(formula,data,distr=mods,method="hmc")
```

In this case, **survHE** will perform the following steps:

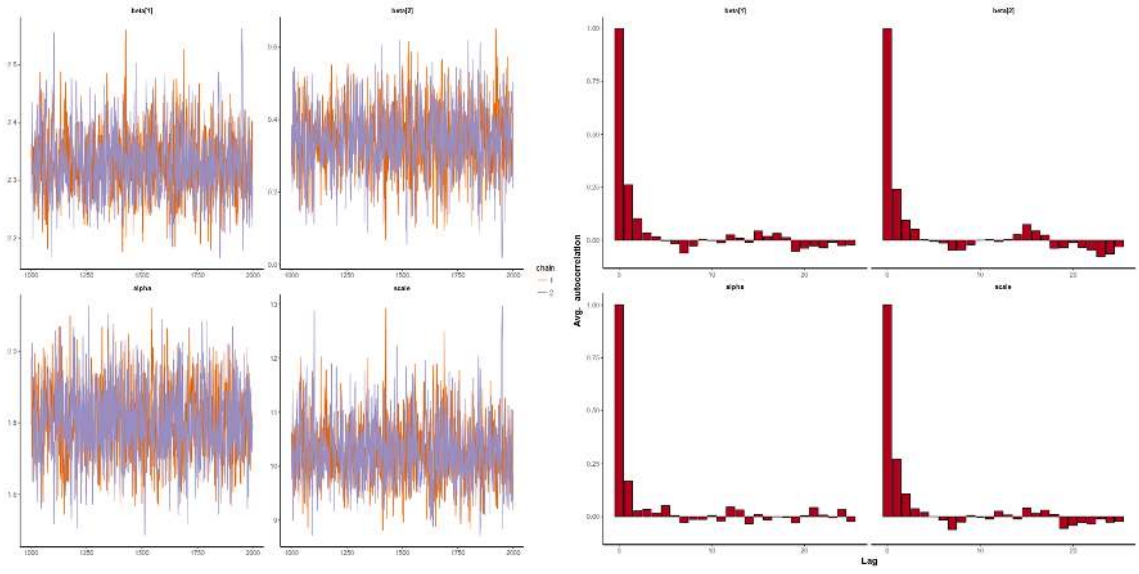
[1] In fact, the relevant target surface in HMC is described by $-\log \tilde{p}(\boldsymbol{\theta} \mid \boldsymbol{t})$, but this is just a technical detail. The general argument still holds.

1. Format the original data contained in the R object `data` in a way that can be used by **rstan**;
2. Selects a pre-compiled model code (depending on the distributional assumptions);
3. Sample from the posterior distribution of the model parameters.

As with any MCMC estimation, it is important to thoroughly assess convergence — we return to this point in §6.1. The package **rstan** contains specialised functions to visualise the model output and assess convergence. For example the commands[2]

```
> rstan::traceplot(m3$models[[2]])
> rstan::stan_ac(m3$models[[2]])
```

would produce respectively a graph with the "traceplots" for the relevant variables, as shown in Figure 1(a) and an autocorrelation plot, shown in Figure 1(b). In this case, we can be confident that convergence is satisfactorily reached for all the variables monitored, since the traceplots show good mixing in the two chains; autocorrelation does not seem a major problem either, as the level of dependence in consecutive iterations wanes down relatively quickly. Additional model checking tools are also available in the package **shinystan** (Stan Development Team 2016), an add-on to **rstan**, which creates a web-app that the user can access locally through the default web browser.



(a) Traceplots (b) Autocorrelation plots

Figure 1: Checking model convergence using the **rstan** built-in facilities, for example through inspection of the traceplots or the autocorrelation plot

There are several additional options that can be used when the inferential method is specified to `'hmc'`, which we describe in the following.

[2]In order to use the functions in the package **rstan**, the user needs to either load the full package using the R command `library(rstan)`, or to prefix their name with the code `rstan::` to make them available in the current workspace.

- `chains`: the number of chains to run in the HMC (default = 2);
- `iter`: the total number of iterations (default = 2000);
- `warmup`: the number of "warm up" iterations (default = `iter`/2). The warm up is the adaptive phase in which the basic inputs of the HMC procedure are tuned. **rstan** does that automatically and once this stage is completed, the procedure is ready to immediately sample from the posterior distributions of interest;
- `thin`: the number of thinning (default = 1). For example, setting thinning to some value h, consists in selecting for inference every one in h simulations from the posteriors and can sometimes reduce the level of autocorrelation (for an equally large number of iterations used for the final estimation);
- `control`: a list specifying **rstan**-related options, *e.g.* `control=list(adapt_delta=0.85)`, which can be used to tune more finely the acceptance rate in the HMC procedure (the higher this rate, the less likely there are to be numerically unstable simulations);
- `seed`: the random seed (to make things replicable);
- `pars`: a sting vector with the names of the relevant parameters;
- `include`: a logical indicator (if set to `FALSE`, then the parameters specified in `pars` are not saved);
- `priors`: a list (of lists) specifying the values for the parameters of the prior distributions in the models;
- `cores`: the number of CPU (cores) used to run the sampling procedure using **rstan** (default = 1)
- `save.stan`: a logical indicator (default = `FALSE`). If `TRUE`, then saves the model text file(s) to the user's working directory. These can be used as template to modify the basic model structure.

In practice, the user should not need to fiddle much with these optional arguments — certainly not without a clear understanding of the underlying modelling assumptions and the implications of any change to the default structure. Perhaps the default number of chains or iteration may be increased; or may be specific numeric values for the parameters of the prior distributions could be defined.

For instance, the default prior for the linear predictor coefficients is $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_J) \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$, where $\boldsymbol{\mu}_\beta$ and $\boldsymbol{\sigma}_\beta$ are vectors of size $(J+1)$ and I_{J+1} is the $(J+1)\times(J+1)$ identity matrix (see Table 2). Suppose the user wanted to select a smaller standard deviation for the Generalised Gamma model; this can be done using the following command[3]

```
> m4=fit.models(formula,data,distr="gengamma",method="hmc",
+      priors=list(list(sigma_beta=rep(5,2)))
)
```

[3]It is worth mentioning that, unlike `OpenBUGS` or `JAGS` (which use the mean and precision), **rstan** parametrises the Normal distribution in terms of the mean and the standard deviation.

— note that we need to specify the values for the standard deviation for all the $J = 2$ covariates (the intercept and the treatment arm) and so in this case we define `sigma_beta=rep(5,2)`, *i.e.* a vector of two elements, each equal to 5. Of course, there is nothing special about the value 5 and we could also select different values for the intercept and the treatment arm, *e.g.* `sigma_beta=c(5,10)`.

It is also possible to specify multiple values to modify the priors, for example

```
> priors=list(list(sigma_beta=rep(4,2)),list(mu_beta=rep(2,2)))
> m4=fit.models(formula,data,distr=mods,method="hmc",priors=priors)
```

would instruct **survHE** that the user wants to set: (a) the value for the standard deviation of the parameters $\boldsymbol{\beta}$ to 4, in the first model to be considered (Exponential); and (b) the the value for the mean of the parameters $\boldsymbol{\beta}$ to 2, for the second model (Weibull).

Because the number of models in `mods` is 6, then **survHE** will complete the list `priors` by adding 4 more more empty lists and **survHE** will use the default values for the remaining models. Consequently, it is important that the user specifies the required values in the correct order. For instance, if we wanted to specify $\sigma \sim \text{Gamma}(2, 4)$ for the Generalised Gamma model (the sixth in the list `mods`), we would need to make sure that this information is contained in the sixth element of the list `priors`. This could be done by using the following code

```
> priors=list()
> for (i in 1:5) {
+ priors[[i]] = list()
}
> priors[[6]] = list(a_sigma=2,b_sigma=4)
```

which creates 5 empty lists to be associated with the first five models and the required list of values for the sixth one. Even more succintly, the same goal can be achieved by typing the following code[4].

```
> priors=replicate(5,list())
> priors[[6]] = list(a_sigma=2,b_sigma=4)
```

Then we can run **survHE** with the same command as before.

```
> m4=fit.models(formula,data,distr=mods,method="hmc",priors=priors)
```

Table 2 shows a summary of the distributional assumptions used to define the default priors in the models implemented by **survHE** using **rstan**, together with the names assigned to the parameters of these distributions. For instance, if we wanted to specify a $\text{Normal}(1, 4)$ prior for the ancillary parameter of the Gompertz model (the fourth in the vector `mods`), we would need to specify the following command.

```
> priors = replicate(6,list())
> priors[[4]] = list(mu_alpha=1,sigma_alpha=4)
```

[4]We note however that, in general terms, when it is necessary to specify complex options (such as the definition of the priors), it is perhaps a better idea to use one single distribution is used in the call to `fit.models`.

Model	Location parameters	Ancillary parameters	Natural parameters
Exponential	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(5,J+1)`	— — —	Rate = $\exp(\beta_0)$
Weibull	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(5,J+1)`	$\alpha \sim \text{Gamma}(a, b)$ a = `a_alpha` $= 0.1$ b = `b_alpha` $= 0.1$	Shape = α Scale = $\exp(\beta_0)$
log-Normal	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(100,J+1)`	$\alpha \sim \text{Uniform}(a, b)$ a = `a_alpha` $= 0$ b = `b_alpha` $= 5$	Mean = β_0 Std. dev. = α
log-Logistic	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(5,J+1)`	$\alpha \sim \text{Gamma}(a, b)$ a = `a_alpha` $= 0.1$ b = `b_alpha` $= 0.1$	Shape = α Rate = $\exp(\beta_0)$
Gamma	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(5,J+1)`	$\alpha \sim \text{Gamma}(a, b)$ a = `a_alpha` $= 0.1$ b = `b_alpha` $= 0.1$	Shape = α Rate = $\exp(\beta_0)$
Gompertz	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(5,J+1)`	$\alpha \sim \text{Normal}(a, b)$ a = `mu_alpha` $= 0$ b = `sigma_alpha` $= 5$	Shape = α Rate = $\exp(\beta_0)$
Gen. Gamma	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$ = `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$ = `sigma_beta=rep(100,J+1)`	$\sigma \sim \text{Gamma}(a_1, b_1)$ $q \sim \text{Normal}(a_2, b_2)$ a_1 = `a_sigma` $= 0.1$ b_1 = `b_sigma` $= 0.1$ a_2 = `mu_Q` $= 0$ b_2 = `sigma_Q` $= 100$	Location = β_0 Scale = σ Shape = q
Gen. F	$\boldsymbol{\beta} \sim \text{Normal}(\boldsymbol{\mu}_\beta, \boldsymbol{\sigma}_\beta I_{J+1})$ $\boldsymbol{\mu}_\beta$= `mu_beta=rep(0,J+1)` $\boldsymbol{\sigma}_\beta$= `sigma_beta=rep(100,J+1)`	$\sigma \sim \text{Gamma}(a_1, b_1)$ $\log(p) \sim \text{Normal}(a_2, b_2)$ $q \sim \text{Normal}(a_2, b_2)$ a_1 = `a_sigma` $= 0.1$ b_1 = `b_sigma` $= 0.1$ a_2 = `mu_P` $= 0$ b_2 = `sigma_P` $= 0.5$ a_3 = `mu_Q` $= 0$ b_3 = `sigma_Q` $= 2.5$	Location = β_0 Scale = σ Shape (1) = q Shape (2) = p

Table 2: A summary of the default assumptions used for the models defined by **survHE** using **rstan**

If the option `save.stan` is set to `TRUE`, then **survHE** will also save the model code as a text file (with the extension `.stan`) in the current directory. The data list formatted in a way that **rstan** can use is also automatically stored in the element `$misc$data.stan` inside the output of `fit.models`. The user can then modify the model structure starting from this template — for example it is possible to change the distributional assumptions and use, *e.g.*, a Uniform prior for the scale σ of a Generalised F model. This will require a new compilation and the new model has to be run using **rstan** commands directly.

6. Summarising the results from survHE

Objects in the class `survHE` (such as `m1`, `m2`, `m3` and `m4` above) can access methods such as `print`, `summary` and `plot` that can be used to summarise and visually inspect the results of the models analysed. We describe them in the following.

6.1. Tabular form

When the models have been estimated, we usually want to summarise the estimates using a tabular format. **survHE** has a specialised function `print` that can do this, *e.g.* by typing

```
> print(m1)
```

which returns the following table.

```
Model fit for the Exponential model, obtained using Flexsurvreg
(Maximum Likelihood Estimate). Running time: 0.009 seconds

                      mean         se       L95%      U95%
rate              0.0824203 0.00828355  0.0676839  0.100365
as.factor(arm)1 -0.4656075 0.15427131 -0.7679738 -0.163241

Model fitting summaries
Akaike Information Criterion (AIC)....: 1274.576
Bayesian Information Criterion (BIC)..: 1282.387
```

In this case, the object `m1` contains many models; but unless the user specifies which one to print, **survHE** will assume that the first one should be used. If for example, we wanted to visualise the estimates for the log-Logistic model (the fifth element of the string vector `distr`), then we would need to type

```
> print(m1,mod=5)
```

which would return the following output.

```
Model fit for the log-Logistic model, obtained using Flexsurvreg
(Maximum Likelihood Estimate). Running time: 0.051 seconds

                     mean        se     L95%    U95%
shape            2.233748 0.1406365 1.974434 2.52712
scale            8.160865 0.5264208 7.191658 9.26069
as.factor(arm)1 0.348356 0.0943506 0.163432 0.53328

Model fitting summaries
Akaike Information Criterion (AIC)....: 1208.494
Bayesian Information Criterion (BIC)..: 1220.211
```

In both cases, **survHE** standardises the format of the output, so that the results are reported for the "basic" parameters (*e.g.* rate, shape or scale) as well as the covariates effects. Notice that the "basic" parameters are always reported on the natural scale, while the covariates effects are in the scale defined by the linear predictor (as presented in Table 1). Thus, in the cases presented above, the value of the coefficient `as.factor(arm)1` represents the impact of the treatment arm on the log scale, because both for the Exponential and the log-Logistic the location parameter is modelled using a log link — and thus in Table 1 we write $\mu_i = \exp(\ldots)$.

The **survHE** method `print` has an additional option, which allows the user to visualise the summary of the model results in the original notation used by the relevant package used to perform the estimation. Thus, in this case typing

```
> print(m1,mod=6,original=TRUE)
```

would show the results for the Generalised Gamma model (the sixth in the string vector `mods`) using the original **flexsurv** formatting.

```
Call:
flexsurv::flexsurvreg(formula = formula, data = data, dist = distr)

Estimates:
                data mean  est     L95%    U95%    se    exp(est)  L95%    U95%
mu                  NA    2.2918  2.1357  2.4479  0.0796     NA      NA      NA
sigma               NA    0.5871  0.4613  0.7471  0.0722     NA      NA      NA
Q                   NA    0.8507  0.3620  1.3394  0.2493     NA      NA      NA
as.factor(arm)1 0.4850    0.3463  0.1743  0.5183  0.0878 1.4138  1.1904  1.6792

N = 367,  Events: 172,  Censored: 195
Total time at risk: 2612.07
Log-likelihood = -598.3923, df = 4
AIC = 1204.785
```

The same principles apply to `survHE` objects storing models fitted using either **INLA** or **rstan**. For instance the command

```
> print(m3,6)
```

returns the output

```
Model fit for the GenGamma model, obtained using Stan (Bayesian inference via
Hamiltonian Monte Carlo). Running time: 42.361 seconds

                    mean        se     L95%     U95%
mu              2.293168 0.0848325 2.127325 2.452178
sigma           0.603824 0.0774517 0.465092 0.763628
Q               0.829105 0.2641518 0.337638 1.360960
as.factor(arm)1 0.344352 0.0930015 0.162244 0.526708

Model fitting summaries
Akaike Information Criterion (AIC)....: 1209.048
Bayesian Information Criterion (BIC)..: 1224.872
Deviance Information Criterion (DIC)..: 1204.997
```

For HMC models, even more importantly, when using the option `original=TRUE` we are able to look at helpful convergence statistics, which should be used to assess whether the MCMC procedure has been successful in exploring the relevant posterior distributions. For example, we could use the code

```
> print(m3,mod=2,original=TRUE)
```

which returns the following output.

```
Inference for Stan model: WeibullAF.
2 chains, each with iter=2000; warmup=1000; thin=1;
```

```
post-warmup draws per chain=1000, total post-warmup draws=2000.

            mean se_mean      sd    2.5%    97.5% n_eff    Rhat
beta[1]  2.33073 0.00149 0.05214 2.23019  2.43722  1227 1.00197
beta[2]  0.34504 0.00261 0.08030 0.18907  0.50086   947 1.00563
alpha    1.79961 0.00278 0.10315 1.59378  2.00339  1380 1.00062
scale   10.29943 0.01545 0.53901 9.30161 11.44115  1217 1.00185

Samples were drawn using NUTS(diag_e) at Tue Nov  8 13:39:35 2016.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

The table displays the original output from `Stan`. Here, the coefficient `beta[1]` is the intercept (log scale), while the coefficient `beta[2]` is the effect of the only covariate included in the model. The estimates are reported as the mean, Monte Carlo error (`se_mean`), standard deviation, an approximate 95% credible interval and then the *effective sample size* `n_eff` and the *Potential Scale Reduction* (PSR) `Rhat`.

The effective sample size gives an indication of the underlying autocorrelation in the MCMC samples — values close to the total number of iterations, or at any rate not too low, indicate a low level of autocorrelation (which is what we want). The PSR is an analysis-of-variance-type of statistics, indicating for each variables whether convergence is reached. If the procedure is ran on more than one parallel chain, then `Rhat` is computed as a function of the ratio of the variance within to the variance between chains — if this is close to 1 and definitely less than 1.1, convergence can be satisfactorily declared. If not, it may be necessary for example to increase the number of iterations.

Further optional inputs to the `print` methods are `digits` (default = 6), which determines the number of digits printed in the table; and `nsim`, the number of MC simulations used to compute the interval estimation for the **INLA** procedure (default = 100)[5].

6.2. Visual interpretation

Once an object of the class `survHE` has been created using the command `fit.models`, it is possible to visualise the resulting survival curve(s) by simply using the `plot` method. This command works irrespective of the underlying inferential engine. For example, the command

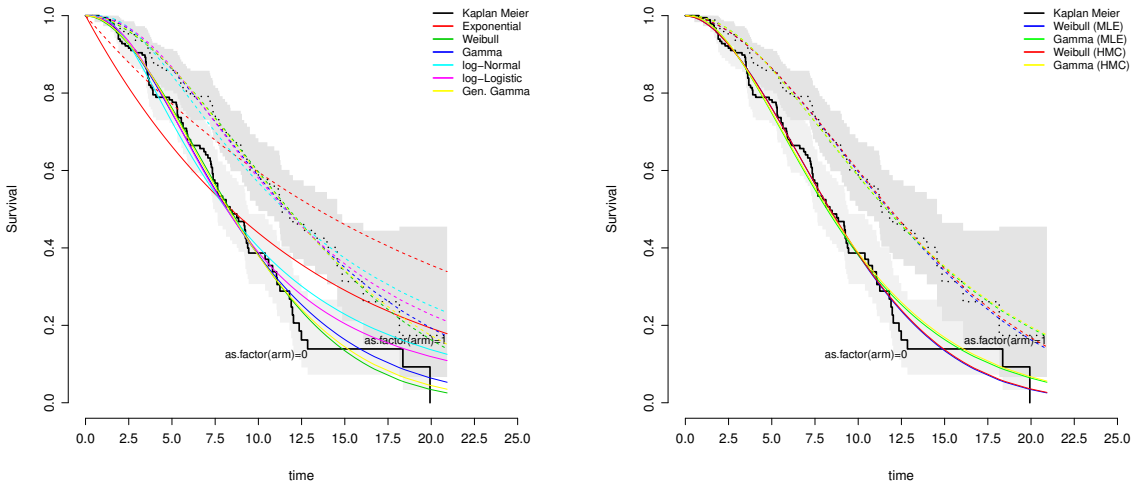
```
> plot(m1)
```

displays the graph shown in Figure 2(a).

The `plot` method allows to display the output from different models. For example, we may be interested in comparing the survival curves from some of the models obtained using either MLE or HMC. We could do this by using the following command.

```
> plot(m1,m3,mod=c(2,3,8,9),colors=c("blue","green","red","yellow"),
+      labs=c("Weibull (MLE)","Gamma (MLE)","Weibull (HMC)","Gamma (HMC)")
)
```

[5]Technically, **INLA** estimates the *marginal* distributions of the model parameters. The joint posterior can then be obtained by a simulation approach using the **INLA** function `inla.posterior.sample`.



(a) Survival curves for the models fitted in the object `m1`

(b) Survival curves for some of the models fitted in the objects `m1` and `m3`

Figure 2: Graphs produced by the `plot` method in **survHE**

This command instructs **survHE** to stack together the objects `m1` and `m3`, which basically produces a single `survHE` object containing 12 models (6 from `m1` and 6 from `m3`). The option `mod=c(2,3,8,9)` selects the models in positions 2, 3, 8 and 9 (*i.e.* the second and third from `m1` and then the second and third from `m3`). The option `colors` can be used to select the colours with which to plot each curve on the graph and similarly the option `labs` overwrites **survHE** default setting and writes a specific text in the label. The resulting graph is shown in Figure 2(b).

The full list of the options for the method `plot` in **survHE** is given below.

- `main`: a string specifying the title of the plot (default at `NULL`);
- `xlab`: a string specifying the label to print on the x-axis of the graph (default = `"time"`);
- `ylab`: a string specifying the label to print on the y-axis of the graph (default = `"Survival"`);
- `lab.trt`: a (vector of) string(s) indicating the labels associated with the strata defining the different survival curves to plot. Default to the value used by the Kaplan-Meier estimate given by the call to `fit.models`;
- `cex.trt`: the factor by which the size of the font used to write the strata is resized (default = 0.8);
- `n.risk`: a logical variable. If `TRUE` (defaults) writes the number at risk at different time points (as determined by the Kaplan-Meier estimate);

- `newdata`: a list (of lists) providing the values for the relevant covariates to stratify the survival curves. If `NULL`, then **survHE** will use the mean value for all the covariates if at least one is a continuous variable, or the full combination of the categorical covariates;
- `xlim`: a vector determining the limits for the x-axis;
- `colors`: a vector of characters defining the colours in which to plot the different survival curves ;
- `labs`: a vector of characters defining the names of the models fitted;
- `add.km`: a logical variable. If `TRUE` (the default value), then also add the Kaplan-Meier estimates of the data to the plot.

Most of these options are actually trivial; perhaps only `newdata` deserves a more detailed explanation. **survHE** follows the philosophy of **flexsurv** to construct the survival curves. Consequently, when the model contains categorical covariates, a single survival curve is estimated for each combination of their modalities. Consider for example the following command.

```
> m5 = fit.models(Surv(time,censored)~as.factor(arm)+as.factor(sex),
+       data=data,distr="exp"
)
```

This estimates the survival times via MLE using an Exponential model and controlling for the effect of the treatment arm and the individuals' sex. The estimates are as follows.

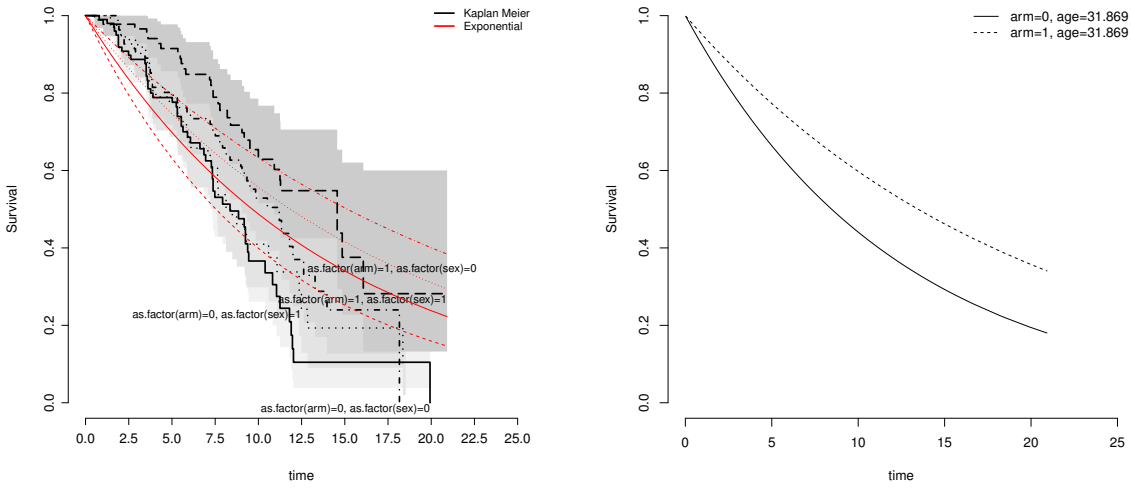
```
Model fit for the Exponential model, obtained using Flexsurvreg
(Maximum Likelihood Estimate). Running time: 0.011 seconds

                       mean        se       L95%       U95%
rate              0.0918686 0.0109122  0.0727882  0.1159506
as.factor(arm)1 -0.4509553 0.1545386 -0.7538454 -0.1480651
as.factor(sex)1 -0.2460077 0.1540280 -0.5478969  0.0558816

Model fitting summaries
Akaike Information Criterion (AIC)....: 1274.005
Bayesian Information Criterion (BIC)..: 1285.721
```

As shown in Figure 3(a), this simple model already becomes complex to visualise, because there are 4 strata identified by the two covariates, each of which is binary. Thus, it may be helpful then to plot the results using a different strategy. For example we can define a list in which we specify the value for the covariates that we want to use to compute the survival curves, *e.g.*

```
> newdata = list(list(arm=0,sex=mean(data$sex)),list(arm=1,sex=mean(data$sex)))
```

which would create two "profiles" by varying the treatment arm and keeping the value for sex to the observed average in the data. The resulting plot can be obtained using the following code.

```
> plot(m5,newdata=newdata)
```



(a) Survival curves for the Exponential model fitted using MLE and controlling for sex and treatment arm

(b) Survival curves for specific values of the covariates, as selected in the list `newdata`

Figure 3: Graphs produced by the `plot` method in **survHE** when using different combination of the covariates values

which produces the graph in Figure 3(b). When the option `newdata` is used, it is probably best to plot a single model in a graph — even if the `survHE` object contains many (just like `m1`, `m2` or `m3`), the option `mod` can be used to select only one of them, as showed above. When selecting the "profile" to be specified in the list `newdata`, the order in which the covariates are entered must be the same in which they appear in the model formula.

6.3. Estimation of the mean survival time

As mentioned earlier, in a health economic evaluation it is of interest to estimate the mean survival time — this is the quantity that is relevant to determine for example the effectiveness of a given intervention. For some of the parametric models described above, the mean survival time can be computed analytically. For example, for a Weibull model, this is

$$\mathrm{E}[T] = \int_0^\infty t f(t \mid \boldsymbol{\theta}) \mathrm{d}t = \int_0^\infty S(t) \mathrm{d}t = \mu \Gamma\left(1 + \frac{1}{\alpha}\right),$$

where μ is the scale, α is the shape and $\Gamma(\cdot)$ is the Gamma function.

More generally, it is possible to approximate this quantity using the "trapezium rule" over a large enough time horizon as

$$\mathrm{E}[T] \approx \frac{1}{2} \sum_{t \in \mathcal{T}} \tau \left[S(t+\tau) - S(t) \right],$$

where $\mathcal{T}$ is a discrete set of $(K+1)$ time points $t = 0, (t+\tau), (t+2\tau), \ldots, (t+K\tau)$ and τ is an arbitrarily small increment. Notice that in order to approximate the mean sufficiently well, it is important to extend the range $\mathcal{T}$ long enough so that all the survival curves actually fade

out to 0. Also, the smaller the increment τ, the more trapezoids are fitted under the survival curve and thus the better the approximation. In practice, there is a trade-off between the level of approximation and the computational time required for the calculation.

survHE automatically performs this calculation by means of the method `summary`; so for example, the R code

```
> summary(m1)
```

produces the following output.

```
Estimated average survival time distribution*
                       mean        sd      2.5%   median     97.5%
as.factor(arm)1=0  7.908986 0.8855609  6.120897  7.90922  9.656117
as.factor(arm)1=1 11.684795 0.8513160 10.003279 11.69654 13.298515

*Computed over the range: [0.03-20.92] using 1000 simulations.
NB: Check that the survival curves tend to 0 over this range!
```

Because the user has not specified a time range over which to compute the mean survival, **survHE** assumes the observed range of times (in this case $[0.03 - 20.92]$). As is obvious from Figure 2, in this range the survival curves have not reached 0 and thus the estimated mean survival is certainly biased downwardly. To correct this, it is sufficient to use the code

```
> summary(m1,t=seq(0,60))
```

which instructs **survHE** to compute the means over a range of times between 0 and 60, with default unit increments. The resulting values are substantially different.

```
Estimated average survival time distribution*
                      mean       sd     2.5%   median    97.5%
as.factor(arm)1=0 12.11621 1.205998 10.05541 11.96503 14.70078
as.factor(arm)1=1 18.45192 1.822470 15.01798 18.44269 22.27354

*Computed over the range: [0-60] using 1000 simulations.
NB: Check that the survival curves tend to 0 over this range!
```

Incidentally, the analytic average values are 12.16317 and 19.72865 for the control and treatment arm, respectively. The estimate produced by the `summary` command can be improved by selecting a longer time horizon and/or a lower value for the increment τ, *e.g.* `t=seq(0,100,0.1)`. It is also possible to increase the number of simulations used to characterised uncertainty in the underlying parameters (and hence survival curves). The default value of 1000 can be modified by specifying the optional argument `nsim` to a different value. Similarly, it is possible to specify a specific "profile", in terms of the covariates included in the model by using the optional argument `newdata`. So for example, the command

```
> summary(m1,t=seq(0,60),newdata=list(list(arm=1)))
```

produces an estimate of the distribution of the mean survival time for the treated. Finally, the user can also include a vector `labs` in the call to `summary`, which contains a suitable

number of text strings that can be used to label the values in the resulting table, for example `labs=c("Controls","Treated")`.

6.4. Model assessment

Health economics guidelines suggest that the assessment of the models is done by complementing the visual inspection with some more formal testing. The suggested way is through the use of a specific Information Criterion (IC), such as Akaike IC (AIC) or the Bayesian IC (BIC). These statistics are based on the model deviance $-2\log\mathcal{L}(\boldsymbol{\theta})$ and a penalty function, which typically depends on the number of parameters and the complexity of the model — the rationale being that models that are too complex tend to "overfit" the data; this means that they may do very well at estimating the data at hand, but usually have poor predictive ability of other data.

An additional IC which is specific to Bayesian models is the Deviance IC (DIC), proposed by Spiegelhalter et al. (2002)[6]. **survHE** then computes an estimate of AIC and BIC for any inference engine and also an estimate of the DIC for models fitted in either **INLA** or **rstan**[7].

The results of model fit can be visually inspected using the **survHE** function `model.fit.plot` as in the examples below.

```
> model.fit.plot(m1)
> model.fit.plot(m1,type="bic")
> model.fit.plot(m1,m3,mod=c(1,2,3,7,8,9),type="dic",
+       mar=c(4,7,3,0.5),xlim=c(1200,1290),
+       models=c("Exponential (MLE)","Weibull (MLE)","Gamma (MLE)",
+              "Exponential (HMC)","Weibull (HMC)","Gamma (HMC)")
)
```

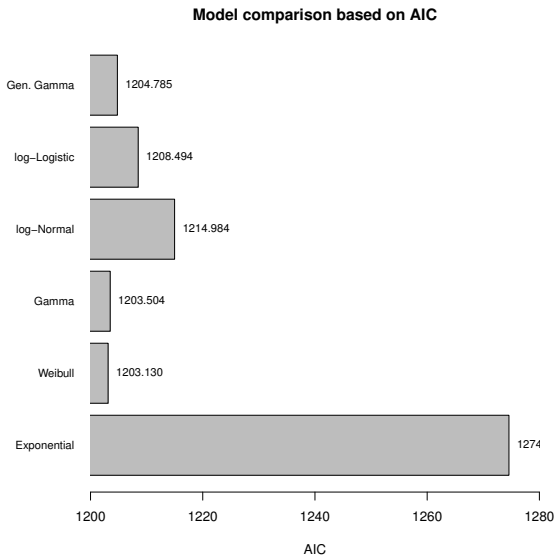
Notice that in panel (c) of Figure 4 for some of the models the bar is not plotted: this is because for model `m1` the DIC cannot be computed, as it is fitted using MLE.

There are many optional arguments to the `model.fit.plot` function:

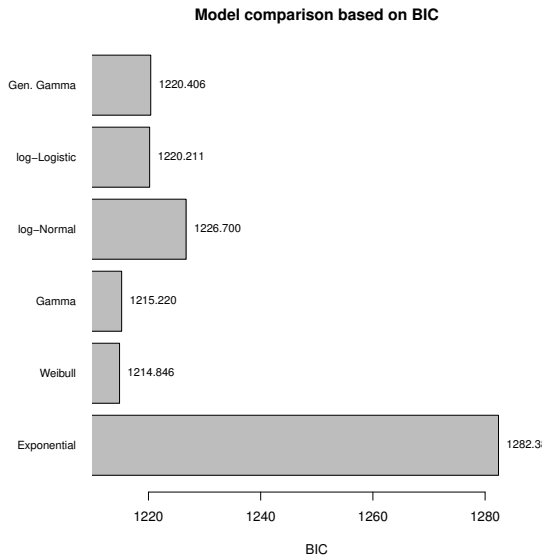
- `type`: a string specifying the statistic to be used (possible values are `'aic'`, `'bic'` or `'dic'`);
- `xlim`: a vector determining the limits for the x-axis;
- `digits`: the number of digits to print next to each bar;
- `main`: the title of the graph;

[6] In fact, for Bayesian models obtained via HMC, **survHE** also computes the DIC using the slightly different definition suggested by Gelman et al. (2013); the two versions of the DIC may differ in the presence of extreme asymmetry or multi-modality in the posterior distributions. The alternative estimates for the DIC are stored in the element `$model.fitting$dic2` of a `survHE` object, but are only present if the inferential method is set to `"hmc"`.

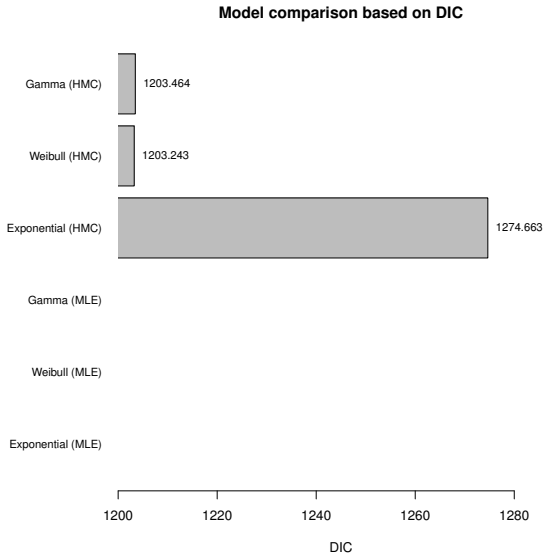
[7] Notice that **rstan** does not provide measures of model fit based on Information Criteria. There are several arguments to prefer other methods to assess the performance of a statistical model, for example the posterior predictive check, as discussed in Gelman et al. (2013). However, because NICE's guidelines suggest using AIC and BIC (and, by extension, DIC), **survHE** computes the relevant model fit statistics and reports them using the `print` method and the `model.fit.plot` function.



(a) Model fit for `m1` using the AIC (the default statistic)



(b) Model fit for `m1` using the BIC



(c) Model fit for selected models in `m1` and `m3`, using the DIC

Figure 4: Example of plots to assess model fit for **survHE** objects

- `mar`: a vector specifying the margins of the R graph. The default value is `c(4,6,3,1.3)`;
- `cex.names`: the factor by which the size of the font used to write text on the graph is resized (default = 0.8);
- `models`: a string of text specifying the names for the models (on the y-axis).

7. Probabilistic sensitivity analysis

Unlike standard epidemiological analysis where the objective is often to estimate the effect of some relevant covariates on the survival time, in health economic evaluation the goal is rather to produce an estimate of the entire survival curve over a long period of time (or at any rate, a longer period than the observed follow up). This estimate is then used to populate the economic model, *e.g.* by obtaining estimates of the transition probabilities between states in a Markov model. More importantly, we need to quantify the impact of uncertainty in the model parameters on the decision-making process and thus we typically repeat this exercise for a large number of times, upon varying the value of the parameters that determine the survival curves. In a fully Bayesian approach this uncertainty is induced by the full joint posterior distribution of the parameters.

survHE is designed to perform this task directly in R, through the function `make.surv`. A typical call is as follows

```
> psa = make.surv(fit=m3,nsim=1000,t=seq(.1,63))
```

which generates an object `psa` containing, among other things, `nsim` = 1000 simulations for the survival curves. In the above case, `m3` contains in fact 6 different models (upon varying the distributional assumptions), but because the user has not specified a value for the input `mod` (in this case a number between 1 and 6), `make.surv` uses the first one, *i.e.* the Exponential, by default. Adding the option `mod=6` to the call to `make.surv` would consider the sixth model (Generalised Gamma) instead.

The resulting output `psa` can be accessed directly by the user. The command

```
> names(psa)
```

shows that it comprises several elements

```
[1] "S"       "sim"     "nsim"    "mat"     "des.mat"
```

each of which can be accessed using either the "double bracket" or the "dollar" notation in R *e.g.* `psa$S` or `psa[[3]]`. A brief description of each of these elements is in the following.

- `S`: a list — for each simulated value of the parameters, a list with the survival curves associated with the configuration of the covariates;
- `sim`: simulated values for the main parameters (*e.g.* scale, shape, rate, mean, sd) for each configuration of the covariates;
- `nsim`: the number of simulations saved;
- `mat`: a list — for each configuration of covariates a matrix with `nsim` rows and as many columns as time points with the survival curves (to be read row-wise);
- `des.mat`: a design matrix with the combination of the covariates used (each represents an element in the lists `S` and `mat`).

The reason why `make.surv` creates so many outputs is mainly for internal convenience. In fact, this is a central function in how **survHE** works and it is called internally by other utility functions (*e.g.* `plot` and `print`). By and large, the user does not need to manipulate the output directly.

The list of inputs for the `make.surv` function is as follows.

- `fit`: the result of the call to the `fit.models` function, containing the model fitting (and other relevant information), *e.g.* `m3` from above;
- `mod`: the index of the model. The default value is 1, but the user can choose which model to visualise, if the call to `fit.models` has a vector argument for the input `distr`;
- `t`: the vector of times to be used to create the survival curves. By default, **survHE** uses the times observed in the original data, but the user can specify a longer follow up (this is actually a useful feature for the purpose of performing a health economic evaluation and PSA);
- `newdata`: a list (of lists), specifiying the values of the covariates at which the computation is performed. For example `'list(list(arm=0),list(arm=1))'` will create two survival curves, one obtained by setting the covariate `'arm'` to the value 0 and the other by setting it to the value 1. In line with **flexsurv** notation, the user needs to either specify the value for *all* the covariates or for none (in which case, `newdata` is set to `'NULL'`, which is the default). If some value is specified and at least one of the covariates is continuous, then a single survival curve will be computed in correspondence of the average values of all the covariates (including the factors, which in this case are expanded into indicators);
- `nsim`: the number of simulations from the distribution of the survival curves. The default is at `nsim` $= 1$, in which case uses the point estimate for the relevant distributional parameters and computes the resulting "average" survival curve.

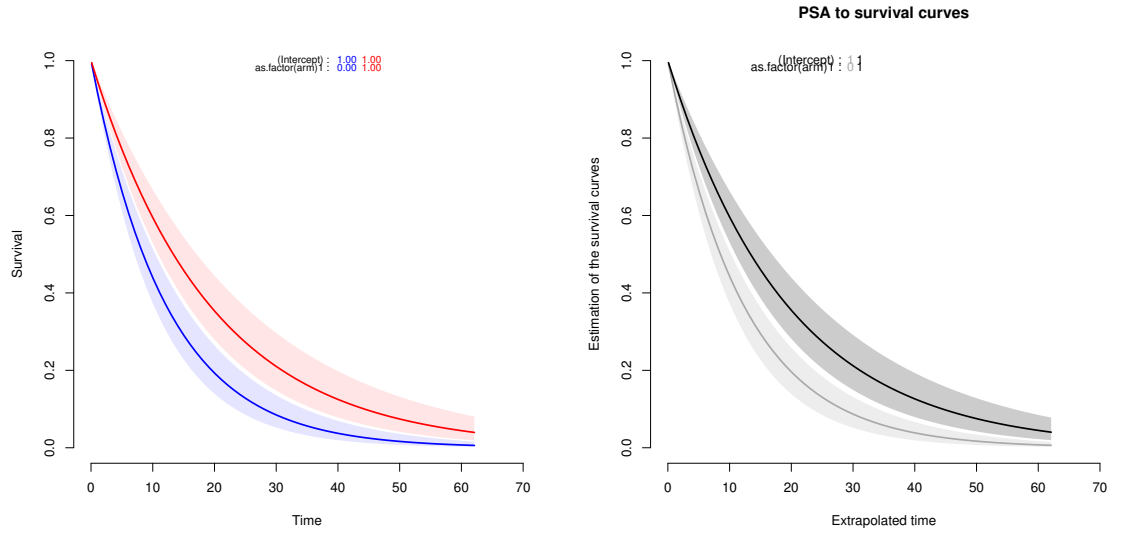
To visualise the results of the PSA, **survHE** has a specialised function called `psa.plot`, which can be used, for example as follows.

```
> psa.plot(psa)
```

This command produces the graph in Figure 5(a). The graph shows the average survival curve and 95% interval estimates around them. If the `method` is set to `'mle'`, then the intervals are obtained by multivariate Normal bootstrap, while for the Bayesian models they are obtained using samples from the relevant posterior distributions.

The user can specify the labels for the x- and y-axis (respectively by including the additional arguments `xlab="..."` and `ylab="..."`). In addition, if no colour specification is given by the user, *e.g.* in the form `col=c("blue","red")`, then **survHE** will randomly choose a colouring scheme. Finally, the parameter `alpha` (default at 0.1) determines the transparency of the curves; values for `alpha` close to 0 imply greater transparency, while values closer to 1 create a solid plot (with no transparency). A fully customised call to `psa.plot` is as follows and produces the graph in Figure 5(b).

```
> psa.plot(psa,xlab="Extrapolated time",ylab="Estimation of the survival curves",
+      alpha=0.2,col=c("dark grey","black"),main="PSA to survival curves",xpos=30,ypos=1,
+      cex.txt=.95,offset=1.5,nsmall=0,digits=2
)
```



(a) Default graph produced by `psa.plot` (b) Fully customised graph produced by `psa.plot`

Figure 5: Probabilistic sensitivity analysis for the survival curves

Notice that it is possible to control the labelling of the profile of covariates associated with each curve, in terms of the font size and appearance (*e.g.* the number of digits printed and the position along the $x-$ and $y-$ axes of the graph).

7.1. Exporting the results to MS Excel

Once the PSA samples are available for the survival curves, it may be easy to continue building the economic model using R. In fact, this is the strategy that we advocate (Baio and Heath 2016). However, we acknowledge that practitioners use `Excel` to produce the economic assessment. Thus, **survHE** has a specialised function, called `write.surv`, that allows the user to export the simulations for the survival curves to a `.xls(x)` file, so that they can be easily used when constructing *e.g.* a Markov model in `Excel`.

As mentioned above, the user can actually manipulate the output of the call to `make.surv` independently and so in a way bypass `write.surv` entirely. However, **survHE** tries to simplify the work process and can be used as follows.

```
> write.surv(psa,file="temp.xls")
```

which produces the following output in R

```
1000 simulation(s) for the survival curve:
 [[1]] = 1,as.factor(arm)1=0
 [[2]] = 1,as.factor(arm)1=1

 Written to file: temp.xls
```

and creates a spreadsheet containing the relevant simulations. The R output clarifies that the resulting spreadsheet contains two tables: the first one is for the survival curves considering

the treatment arm set to 0 (*e.g.* controls), while the second one is for the intervention arm (set to 1). Different model specifications would create a different number of matrices (with `nsim` rows and as many time points for columns) depending on the covariates combinations.

8. Advanced models

While the models presented in Table 1 are likely to produce at least one "good" candidate in most situations, it is possible that more complex model structures may be needed to accommodate a particular data set or analysis. In particular, we focus here on two "flexible" models: the first one (which is already implemented in **flexsurv**) is based on cubic splines, while the second is an extension of the standard Weibull model.

8.1. Royston-Parmar splines

Splines are numeric functions typically defined as a collection of local polynomials; the main idea is to partition the $x-$axis into a set of intervals defined by "knots" and then within each interval use a different polynomial function to approximate the underlying "true" function on the $y-$axis. This construction provides great flexibility and effectively an arbitrary number of parameters, depending on how many knots (and hence on the density with which the $x-$axis is partitioned) are selected.

In the context of survival analysis, splines can be used to model flexibly (a suitable transformation of) one of the basic functions, *e.g.* the survival or the hazard (Royston and Lambert 2011). An increasingly popular model is the one developed by Royston and Parmar (2002). Basically, this defines a probability distribution to model the observed and censored times as a function of an "augmented" set of parameters $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma})$ and covariates $(\boldsymbol{X}, \boldsymbol{B})$. Here, $\boldsymbol{\beta}$ are the coefficients associated with the observed covariates $\boldsymbol{X}$, exactly as in (2). In addition, for each individual (and, hence, observed time) in the dataset, we consider a set of $(M+2)$ "basis" function $\boldsymbol{B}_i = (B_{i0}, B_{i1}, \ldots, B_{iM})$, where

$$B_{im} = (\log t_i - k_m)_+^3 - \lambda_m (\log t_i - k_{min})_+^3 - (1-\lambda_m)(\log t_i - k_{max})_+^3, \quad \lambda_m = \frac{k_{max} - k_m}{k_{max} - k_{min}}$$

and $(\log t_i - a)_+ := \max\{0, (\log t_i - a)\}$. The vector of knots is defined as $k_{min} = 0 < k_1 < \ldots < k_{max} = \infty$; typically, the M "internal" knots are set up in terms of the quantiles of the observed distribution of the times. For example, if we set $M = 3$, **survHE** would automatically consider the three quartiles (q_1, q_2, q_3), representing the 25%–, 50%– and 75%–percentiles of the observed times distribution.

The Royston-Parmar (RP) model defines a modified linear predictor

$$\eta_i = \sum_{m=0}^{M+2} \gamma_m B_{im} \left[+ \sum_{j=0}^{J} \beta_j X_{ij} \right],$$

which is used to model directly $\log\left(-\log S(t_i)\right) = \log H\left(t_i\right) = \eta_i$ (notice that we use the [+...] notation here to highlight the fact that the model may not include any covariate $\boldsymbol{X}$ and thus only rely on the splines structure)[8].

[8] In fact, there are three different versions of the RP model; the one presented here is the "proportional

Given this set up, it is possible to prove that

$$\log h(t_i) = -\log t_i + \log \eta_i' + \eta_i \tag{4}$$

and

$$\log S(t_i) = -\exp(\eta_i), \tag{5}$$

with $\eta_i' = \sum_{m=1}^{M+2} \gamma_m B_{im}'$ — the first derivatives B_{im}' are easy to compute, recalling that $\frac{\partial x^3}{\partial x} = 3x^2$. Substituting (4) and (5) into (1), we can completely characterise the resulting likelihood.

The RP model is directly available in **flexsurv** and is also implemented under a pre-compiled HMC-based Bayesian framework in **survHE**. The two versions can be obtained via a specialised call to `fit.models`.

```
> formula = Surv(time,censored)~as.factor(arm)
> m6 = fit.models(formula=formula,data=data,distr="rps",k=2)
> m7 = fit.models(formula=formula,data=data,distr="rps",k=2,method="hmc")
```

survHE accepts the string `"rps"` to indicate the "Royston-Parmar Splines" distribution and also requires the input `k` to specify the number M of internal knots to be used (if this is not provided by the user, **survHE** assumes that `k=0`, which reduces the RP model to a Weibull PH formulation). The `formula` is used to specify the "fixed" component of η_i, *i.e.* the set of covariates in $\boldsymbol{X}$. In this case, we use only the treatment arm.

All the usual methods are available for the resulting `survHE` objects `m6` and `m7`, for example the commands

```
> print(m6)
> print(m7)
```

return the summary tables

```
Model fit for the Royston & Parmar splines model, obtained using Flexsurvreg
(Maximum Likelihood Estimate). Running time: 0.145 seconds

                     mean       se      L95%      U95%
gamma0          -4.652836 0.474704 -5.583239 -3.722433
gamma1           2.523687 0.565148  1.416017  3.631357
gamma2           0.387949 0.331049 -0.260894  1.036792
gamma3          -0.419667 0.398441 -1.200596  0.361262
as.factor(arm)1 -0.615885 0.155792 -0.921232 -0.310538

Model fitting summaries
Akaike Information Criterion (AIC)....: 1205.235
Bayesian Information Criterion (BIC)..: 1224.761
```

and

hazard", which can be seen as an extension to the basic Weibull PH model. The other two versions extend the log-Logistic model by setting $\log(1/S(t) - 1) = \eta_i$ ("proportional odds") and the log-Normal model using $\Phi^{-1}(S(t)) = \eta_i$ ("probit model"). Currently, all are implemented in **flexsurv**, while only the PH model is implemented in **survHE** using HMC.

```
Model fit for the Royston & Parmar splines model, obtained using Stan (Bayesian
inference via Hamiltonian Monte Carlo). Running time: 50.217 seconds

                     mean       se      L95%      U95%
gamma0          -4.667948 0.473789 -5.618594 -3.788113
gamma1           2.527583 0.547405  1.475757  3.650847
gamma2           0.386299 0.308786 -0.207955  1.006886
gamma3          -0.417838 0.370723 -1.162944  0.265064
as.factor(arm)1 -0.622932 0.153474 -0.920748 -0.333737

Model fitting summaries
Akaike Information Criterion (AIC)....: 1205.243
Bayesian Information Criterion (BIC)..: 1224.770
Deviance Information Criterion (DIC)..: 1259.883
```

The Bayesian version of the RP model is much more computationally intensive, although HMC does a very good job and keeps the time to generally acceptable levels; also, it helps in this case to take advantage of the multi-processing capability of **rstan**: adding the option `cores=2` reduces the time from 50.217 to 32.777 seconds, for 2 chains of 2000 iterations each.

In this case, there is very good agreement in the point and interval estimates for the two versions of the model, but in general the MLE may underestimate the underlying level of correlation among the $\boldsymbol{\gamma}$ coefficients in particular.

8.2. Poly-Weibull

In a nutshell, the Poly-Weibull model (Berger and Sun 1993; Demiris et al. 2015) extends the basic set up of a Weibull survival model by accounting for the possibility that in fact the observed times are the result of a mixed data generating process, depending on several independent Weibull components. For example, we may consider that the occurrence of the event under study depends on M independent causes and that we are willing to model each using a suitable Weibull distribution. In line with (1), the resulting density is

$$\begin{aligned} f(t_i \mid \boldsymbol{\theta}) &= h(t_i)^{d_i} S(t_i) \\ &= \left[\sum_{m=1}^{M} \alpha_m \mu_{im} t_i^{\alpha_m - 1}\right]^{d_i} \left[\exp\left(-\sum_{m=1}^{M} \mu_{im} t_i^{\alpha_m}\right)\right], \end{aligned}$$

where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_M)$ and $\boldsymbol{\theta}_m = (\alpha_m, \mu_{im})$ are the shape and scale for the $m-$th component of the mixture.

survHE implements this density as an add-on model to be estimated using HMC and **rstan** (*i.e.* there is only a Bayesian version for this model). While it is in principle possible to model both the shape and the scale as functions of a set of covariates, **survHE** considers the simpler version where only the location parameter is allowed to depend on $\boldsymbol{X}$. It is fairly easy to modify this structure and implement a version of the **rstan** model in which also α_m depend on the covariates.

Practically, **survHE** has a specific function to run the Poly-Weibull model. A typical call is as in the following.

```
> formula.pw=list(Surv(time,censored)~1,Surv(time,censored)~as.factor(arm))
> m8 = poly.weibull(formula.pw,data,cores=2)
```

The main difference with respect to the standard call to `fit.models` is that the formula input now needs to be made by a *list* of objects in the class `formula`. This is because we need to specify a formula for each of the components that we want to fit to the mixture model identified by the Poly-Weibull distribution. For instance, in the above example, the object `formula.pw` is a list with two elements. This instructs R to assume a model with $M = 2$ components and the following specification for the linear predictors

$$\mu_{i1} = \exp(\beta_{01}) \qquad \text{and} \qquad \mu_{i2} = \exp(\beta_{02} + \beta_{12}\text{Arm}_i).$$

By default, **survHE** places minimally informative priors on the parameters $\boldsymbol{\beta}_m \overset{iid}{\sim} \text{Normal}(0, 10)$; the values for the mean (`mu_beta`) and the standard deviation (`sigma_beta`) can be modified using the option `prior`, as shown earlier. In addition, we need to impose an identifiability constraint on the shape parameters $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)$ so that they are ordered (*i.e.* $\alpha_1 < \ldots < \alpha_M$) — see Demiris et al. (2015) for a discussion of this issue. The components of the vector $\boldsymbol{\alpha}$ are then given a vague prior over their entire domain.

As is often the case for mixture models, convergence to the posterior distributions may be a crucial issue. This is essentially due to the fact that it may be very difficult (or even impossible) for the model to distinguish two or more of the components. For example, the above call to `poly.weibull` returns the following warning

```
Warning messages:
1: There were 16 divergent transitions after warmup. Increasing adapt_delta above 0.8
may help. See http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
2: Examine the pairs() plot to diagnose sampling problems
```

indicating that the HMC algorigthm has failed to fully explore the posterior density of the parameters. Possible solutions are to either include more information in the priors (*e.g.* by reducing the range of variation of the coefficients in the $\boldsymbol{\beta}_m$), or to increase the value of the acceptance rate (see §5.2). For example, the command

```
> m9 = poly.weibull(formula.pw,data,cores=2,
+       control=list(adapt_delta=.9,stepsize=.01,max_treedepth=100)
)
```

modifies the standard **rstan** settings to have a denser discrete approximation of the continuous Hamiltonian dynamics[9]. This increases the running time from 35.866 to 120.126 seconds, but successfully estimates the posterior distributions. The results can be analysed by using the `print` method.

```
> print(m9)
```

[9] In brief, a single iteration of HMC proceeds to update the value of the parameters for `max_treedepth` steps before deciding whether to accept or reject the current value. Each step is scaled by a factor `stepsize`, which determines the level of the discrete approximation to the underlying continuous Hamiltonian dynamics. Thus, increasing the target acceptance rate `adapt_delta` effectively forces smaller step sizes, meaning that the algorithm will be in principle more thorough in visiting the posterior density (at the expense of computational time). Similarly, increasing `max_treedepth` means that the algorithm will take more samples within each step, thus potentially being more accurate. In practice, it is important to strike a balance and avoid too fine a discretisation of the Hamiltonian dynamics, while guaranteeing sufficient coverage of the posterior density. More technical details can be found in Hoffman and Gelman (2014) and Carpenter et al. (2015).

```
Model fit for the PolyWeibull model, obtained using Stan (Bayesian inference via
Hamiltonian Monte Carlo). Running time: 120.126 seconds

                       mean       se       L95%      U95%
shape_1             1.35248 0.527126   0.114561  1.921400
shape_2             2.02113 0.357700   1.655006  2.796549
(Intercept)_1      -8.12990 5.336881 -22.160829 -4.064798
(Intercept)_2      -5.16950 1.253134  -7.747451 -3.859921
as.factor(arm)1_2 -4.16059 5.112734 -17.966250 -0.358703

Model fitting summaries
Akaike Information Criterion (AIC)....: 1209.618
Bayesian Information Criterion (BIC)..: 1229.145
Deviance Information Criterion (DIC)..: 1202.083
```

The resulting table identifies the parameters of each component by using a suffix "`_m`", so for instance `(Intercept)_1` is the intercept for the first component.

Notice that in this particular case, using a more complex model does not seem to improve things substantially: for example, the DIC for the simpler Weibull model we fitted in the second element of the object `m3` is 1203.243 (cfr. Figure 4c) — essentially the same as for the Poly-Weibull model, indicating that a simpler version is perhaps to be preferred to *this* specification of the more complex one. This is consistent with the potential issues in convergence and identifiability, which again indicate perhaps that the Poly-Weibull model may not be appropriate for this particular set of data.

9. Other tools

9.1. Digitising Kaplan-Meier curves from published studies

Often, the individual level data from, *e.g.*, a clinical trial measuring the survival times are not directly available for the health economic evaluation. Perhaps for one of the treatments being considered, the sponsor of the trial is able to make the data available, but this could only cover one of the relevant interventions/drugs. To overcome this limitation, usually modellers try and use systematic reviews of the available literature to gather information on the possible comparators.

One clever way of doing this is by "digitising" Kaplan-Meier data available from published papers. This is done by using specific software (*e.g.* `DigitizeIt`); the user needs to click on several points on the survival curves and the values are digitised and exported to some output files, describing the input survival times from graph reading and reported number of individuals at risk at several time points in the follow up.

Once these two files are available, **survHE** takes them as input and following the algorightm developed by Guyot et al. (2012), which can be used to map from the digitised curves back to the unobserved Kaplan-Meier data by numerical approximation. Assuming that the `DigitizeIt` outcome is saved in the two files `survival.txt` and `nrisk.txt` in the current working directory, a typical **survHE** call to perform this task is the following.

```
> surv_inp = "survival.txt"
> nrisk_inp = "nrisk.txt"
```

```
> km_out = "KMdata.txt"
> ipd_out "IPDdata.txt"
> digitise(surv_inp=surv_inp,nrisk_inp=nrisk_inp,
+          km_output=km_out,ipd_output=ipd_out
)
```

The above code first defines the input and output files and then uses the **survHE** command `digitise` to reconstruct the original Kaplan-Meier curves and save their values, as well as a fictional dataset, which closely resembles the one that has generated the published survival curves. `digitise` also write to the R terminal the following text, to make sure that the user can retreive the relevant files.

```
Kaplan Meier data written to file: KMdata.txt
IPD data written to file: IPDdata.txt
```

These can be then used as input data to fit the survival models and then perform the PSA.

The final feature worth mentioning is that often we will be in a position of creating several individual level dataset to mimic data originally used to produce Kaplan-Meier estimates for the survival curves, *e.g.* for many different treatments, or for the same treatment observed in several papers. **survHE** has another specialised function that can be used to stack the different files with the individual level data into a single dataset. This function is called `make.ipd` and it takes as inputs: a list of all the the names of the individual level data files created as output of digitise; the index of the file associated with the control arm (by default, this is the first file) and the control arm will be coded as 0; and a vector of labels for the column of the resulting data matrix. These should match the arguments to the formula specified for the function `fit.models` and should be 3 elements (each representing the the time variable, the event variable and the treatment arm). A call to this function would look like the following.

```
> ipd_files = list("IPD1.txt","IPD2.txt","IPD3.txt")
> data = make.ipd(ipd_files,ctr=1,var.labs=c("time","event","arm")) {
```

This generates a R `data.frame`, which can then be fed to the `fit.models` function. Naturally, `make.ipd` assumes that there are no other covariates in addition to the treatment arm, because it is unlikely that digitised data are recorded for different strata, or values of additional variables.

10. Conclusions

The author wishes to thank Peter Konings, William Browne and Andrea Berardi for providing help in writing part of the original code in **survHE**. This work has been partially supported by a research grant sponsored by Mapi.

References

G. Baio. *Bayesian Methods in Health Economics*. Chapman Hall/CRC Press, Boca Raton, FL, 2012.

G. Baio and P. Dawid. Probabilistic Sensitivity Analysis in Health Economics. *Statistical Methods in Medical Research*, doi: 0962280211419832:First published 18 September 2011, 2011.

G. Baio and A. Heath. When simple becomes complicated: Why Excel should lose its place at the top table. *Global & Regional Health Technology Assessment*, 2016.

J. Berger and D. Sun. Bayesian Analysis for the Poly-Weibull Distribution. *Journal of the American Statistical Association*, 88:1412–1418, 1993.

M. Blangiardo and M. Cameletti. *Spatial and Spatio-temporal Bayesian Models with* `R-INLA`. Wiley, 2015.

M. Blangiardo, M. Cameletti, G. Baio, and H. Rue. Spatial and spatio-temporal models with `R-INLA`. *Spat Spatiotemporal Epidemiol*, 7:39–55, Dec 2013.

A. Briggs, M. Sculpher, and K. Claxton. *Decision modelling for health economic evaluation*. Oxford University Press, Oxford, UK, 2006.

B. Carpenter, M. Gelman, A andd Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and R. A. `Stan`: A Probabilistic Programming Language. *Journal of Statistical Software*, 2015.

N. Demiris, D. Lunn, and L. Sharples. Survival extrapolation using the poly-Weibull model. *Statistical Methods in Medical Research*, 24:287–301, 2015.

A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin. *Bayesian Data Analysis - 3rd edition*. Chapman Hall/CRC, New York, NY, 2013.

P. Guyot, A. E. Ades, M. J. Ouwens, and N. J. Welton. Enhanced secondary analysis of survival data: reconstructing the data from published Kaplan-Meier survival curves. *BMC Medical Research Methodology*, 12:9, Feb 2012.

M. Hoffman and A. Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning*, 15:1593–623, 2014.

C. Jackson. `flexsurv`: A Platform for Parametric Survival Modelling in R. *Journal of Statistical Software*, 2016. doi:10.18637/jss.v070.i08.

N. Latimer. NICE DSU Technical Support Document 14. 2011. URL http://www.nicedsu.org.uk.

T. Martins, D. Simpson, F. Lindgren, and H. Rue. Bayesian computing with INLA: New features. *Computational Statistics & Data Analysis*, 67:68–83, 2013.

N. Radford. MCMC Using Hamiltonian Dynamics. In Brooks, S. and Gelman, A. and Jones, G. and Xiao-Li, M., editor, *Handbook of Markov Chain Monte Carlo*. Chapman Hall/CRC Press, Boca Raton, FL, 2011.

P. Royston and P. Lambert. *Flexible Parametric Survival Analysis Using Stata: Beyond the Cox Model*. Stata Press, 2011.

P. Royston and M. Parmar. Flexible Parametric Proportional-Hazards and Proportional Odds Models for Censored Survival Data, with Application to Prognostic Modelling and Estimation of Treatment Effects. *Statistics in Medicine*, 21:2175–2197, 2002.

H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*. Chapman Hall/CRC Press, Boca Raton, FL, 2005.

H. Rue, S. Martino, and N. Chopin. Approximate Bayesian Inference for Latent Gaussian Models Using Integrated Nested Laplace Approximations (with discussion). *Journal of the Royal Statistical Society* B, 71:319–392, 2009.

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

`Stan` Development Team. `ShinyStan`: Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models. 2016. URL `http://mc-stan.org`.

A. Willan and A. Briggs. *The statistical analysis of cost-effectiveness data*. John Wiley and Sons, Chichester, UK, 2006.

C. Williams, J. Lewsey, A. Briggs, and D. Mackay. Estimation of survival probabilities for use in cost-effectiveness analysis: a comparison of a multi-state modelling survival analysis approach with partitioned survival and Markov decision-analytic modelling. *Medical Decision Making*, 2016.

Affiliation:

Gianluca Baio
Department of Statistical Science
University College London
Gower Street, London, WC1E 6BT (UK)
E-mail: `g.baio@ucl.ac.uk`
URL: `http://www.statistica.it/gianluca`