




## pexm: A JAGS Module for Applications Involving the Piecewise Exponential Distribution

Vinícius D. Mayrink 

Universidade Federal  
de Minas Gerais

João Daniel N. Duarte

Universidade Federal  
de Minas Gerais

Fábio N. Demarqui 

Universidade Federal  
de Minas Gerais

---

### Abstract

In this study, we present a new module built for users interested in a programming language similar to BUGS to fit a Bayesian model based on the piecewise exponential (PE) distribution. The module is an extension to the open-source program JAGS by which a Gibbs sampler can be applied without requiring the derivation of complete conditionals and the subsequent implementation of strategies to draw samples from unknown distributions. The PE distribution is widely used in the fields of survival analysis and reliability. Currently, it can only be implemented in JAGS through methods to indirectly specify the likelihood based on the Poisson or Bernoulli probabilities. Our module provides a more straightforward implementation and is thus more attractive to the researchers aiming to spend more time exploring the results from the Bayesian inference rather than implementing the Markov Chain Monte Carlo algorithm. For those interested in extending JAGS, this work can be seen as a tutorial including important information not well investigated or organized in other materials. Here, we describe how to use the module taking advantage of the interface between R and JAGS. A short simulation study is developed to ensure that the module behaves well and a real illustration, involving two PE models, exhibits a context where the module can be used in practice.

*Keywords:* MCMC, Bayesian inference, semiparametric, survival analysis.

---

## 1. Introduction

The piecewise exponential (PE) modeling is a simple approach extensively used in survival analysis and reliability to approximate the distribution of event-time data; possibly investigating the association of the time response with explanatory variables. Applications related to clinical data can be easily found in the literature; for example: leukemia (Breslow 1974), gastric cancer (Gameran 1991), kidney infection (Sahu, Dey, Aslanidou, and

Sinha 1997; Ibrahim, Chen, and Sinha 2001), breast cancer (Sinha, Chen, and Ghosh 1999), melanoma (Demarqui, Dey, Loschi, and Colosimo 2014) and hospital mortality (Clark and Ryan 2002). In the context of reliability, consider the maintenance analysis in Rigdon and Basu (2000) and the telecommunication study in Kim and Proschan (1991) and Gamerman (1994), among many other references.

The PE models are classified as semiparametric due to their flexibility to represent the hazard function without making strong and restrictive parametric assumptions regarding the shape of this component; this point makes the PE distribution very attractive for survival analysis. Its semiparametric nature allows considerable generality and enough structure for useful interpretations in any application. The popularity of semiparametric methods for univariate survival data started with Cox (1972) on the proportional hazards model. Breslow (1972) and Breslow (1974) proposed the use of the PE distribution to replace the baseline term and Kalbfleisch and Prentice (1973) investigated the grid configuration. In this paper, we are focused on PE models under the Bayesian approach for inference, which has been widely considered in the literature. See Ibrahim *et al.* (2001) for a broad presentation of Bayesian survival models and Sinha and Dey (1997) for a review on Bayesian semiparametric frameworks (including the PE) based on either the hazard or the intensity function.

Implementing a PE model is an obstacle for those interested in a Bayesian analysis and having limited programming skills. In many cases, the main source of this difficulty is the use of an indirect method to draw samples from the unknown joint posterior distribution. The Markov Chain Monte Carlo (MCMC) algorithm Gibbs sampling (see Geman and Geman 1984; Gelfand and Smith 1990; Gamerman and Lopes 2006) is widely applied for this task, and its implementation can be very demanding due to the sequential simulations from complete conditional posterior distributions. This is particularly true when the conjugate analysis is not possible leading to a Gibbs sampler whose steps depend on other methods – e.g., Metropolis-Hastings (Metropolis, Rosenbluth, Teller, and Teller 1953; Hastings 1970), adaptive rejection sampling (Gilks and Wild 1992; Gilks 1992), adaptive rejection Metropolis sampling (Gilks, Best, and Tan 1995) and slice sampling (Neal 2003).

As an alternative to simplify the Gibbs sampler implementation for a given model, one can take advantage of programs based on a dialect of the BUGS language (Gilks, Thomas, and Spiegelhalter 1994; Lunn, Jackson, Best, Thomas, and Spiegelhalter 2012) using the mathematical formalism of directed acyclic graphs to define joint densities. These programs build the whole structure of complete conditionals and their corresponding sampling strategies requiring from the user only the essential information related to the model fit (priors, likelihood and the MCMC setup). In this category, WinBUGS (Spiegelhalter, Thomas, and Best 2003), OpenBUGS (Spiegelhalter, Thomas, Best, and Lunn 2014) and JAGS (Plummer 2003) are three well known options. In particular, JAGS is an appealing case since it is open-source (released under a free copyleft license) and extensible allowing users with C++ (Stroustrup 2013) knowledge to create functions, monitors, distributions and samplers for a new module. These features enable the potential formation of a future JAGS contributor community similar to that of the software R (R Core Team 2021).

Other alternatives are available as a support for the Bayesian computing. A well known case is the platform Stan (Carpenter, Gelman, Hoffman, Lee, Goodrich, Betancourt, Brubaker, Guo, Li, and Riddell 2017), having an interface with R through the package `rstan` (Stan Development Team 2020). The MCMC method applied here is called No-U-Turn Sampling (Hoffman and Gelman 2014), being an extension of the Hamiltonian Monte Carlo method. Implement-

ing new distributions in **Stan** is straightforward and it does not require the installation of new modules. However, due to the Hamiltonian dynamics to improve sampling, one cannot set discrete prior distributions. This is perhaps the greatest weakness of **Stan**. Another possibility to deal with Bayesian modeling in R is the package **BayesX** (Brezger, Kneib, and Lang 2005). However, this alternative was designed for the framework of generalized linear models, where the distribution of the response variable belongs to the exponential family. This is not the case for survival models assuming the PE distribution. Bayesian computing can also be developed via INLA (Rue, Martino, and Chopin 2009); further details in [r-inla.org](https://r-inla.org). This option performs approximate Bayesian inference on the class of latent Gaussian models. It handles non-Gaussian likelihoods, but the presence of non-Gaussian latent components can be a limitation; for example, assuming a discrete or a multimodal distribution for a random effect would be problematic.

The main contribution of this work is to present package **pexm**, a new **JAGS** module for Bayesian analyses involving the PE distribution. An R package, having the same name, was developed to install and load the tool. Package **pexm** (Mayrink, Duarte, and Demarqui 2021) is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=pexm>. The proposed module was created to fill a gap existing in **JAGS** related to the absence of the PE distribution. The **pexm** package allows a friendlier and cleaner model implementation, being attractive for many users. Computational speed is also an advantage of the proposal with respect to the usual implementation based on the Poisson zeros-trick. Another important contribution is a description to clarify some aspects related to the topic “extending **JAGS**”, which is not well documented in the literature. There are no formal tutorials exploring the technical elements detailed in Appendix A. The **JAGS** manuals (Plummer 2010, 2017) do not currently contain the mentioned information, however, these materials can be updated in the future.

The outline of this paper is as follows: in Section 2, we describe the elements of the PE distribution implemented in the proposed **JAGS** module. In Section 3, we present an overview on how the module was created, explain how to use it from within R, discuss the main simplifications with respect to an implementation without the module and develop a simulation study to verify the performance. Section 4 shows results (with and without the module) in a real application involving two frailty models to fit a kidney infection data (Ibrahim *et al.* 2001); the rates and log-rates in adjacent intervals are correlated via gamma and normal priors, respectively. Finally, Section 5 presents the conclusions and final remarks.

## 2. The piecewise exponential distribution

The PE distribution requires the specification of a grid  $\tau = \{a_1, a_2, \dots, a_m\}$  partitioning the time axis into  $m$  intervals. Let  $a_1 = 0$ ,  $a_m < +\infty$  and  $a_{m+1} = +\infty$ ; the latter is defined only for notation purposes and it is not expressed in  $\tau$ . Conditional on this time grid, the failure rate  $\lambda_j$  is assumed constant inside the corresponding interval:  $I_j = (a_j, a_{j+1}]$  for  $j = 1, \dots, m-1$  and  $I_m = (a_m, +\infty)$  for  $j = m$ . This step function formulation provides a discrete version of the true unknown hazard function in a survival model. Assume that  $T$  is a continuous non-negative random variable representing the survival times of subjects in a population. Let  $f(t)$  denote the probability density function (pdf) of  $T$ ,  $F(t)$  is the associated cumulative distribution function (cdf),  $S(t) = 1 - F(t)$  is the survival function,  $h(t)$  is the hazard function and, finally,  $H(t)$  represents the cumulative hazard function. If  $T \sim \text{PE}(\lambda, \tau)$

with  $\lambda = \{\lambda_1, \dots, \lambda_m\}$ , then:

$$h(t) = \lambda_j, \text{ if } t \in I_j, \quad j = 1, \dots, m. \quad (1)$$

Naturally, the quality of the approximation to the real hazard function depends on a suitable choice of the time grid  $\tau$ . In [Breslow \(1972\)](#) the author suggests the use of the observed failure times as the limits of the intervals  $I_j$ ; however, depending on the sample size, this can lead to a non-parsimonious model with many rates  $\lambda_j$  to be estimated. In contrast, [Kalbfleisch and Prentice \(1973\)](#) indicate that the grid should be selected independently of the data. The study explores different interval sizes configuring a regular grid. When choosing  $\tau$  without using the data, one might consider two aspects: small  $m$  leads to a poor discretization and a large  $m$  determines a non-parsimonious model. Irregular grids are also allowed in the analysis and this could be a reasonable strategy in applications assuming higher frequency of failures in certain parts of the time axis; i.e., wider intervals would be specified for time regions where few events are expected to be observed.

The cumulative hazard function is expressed as follows:

$$H(t) = \begin{cases} \lambda_1 t, & \text{if } t \in I_1; \\ \lambda_j(t - a_j) + \sum_{i=1}^{j-1} \lambda_i(a_{i+1} - a_i), & \text{if } t \in I_j, \quad j = 2, \dots, m. \end{cases} \quad (2)$$

We can use Equation 2 to evaluate the survival function at a time point  $t$  through the well known equality  $S(t) = \exp\{-H(t)\}$ . The pdf of  $T$  can be easily identified by simply taking the derivative of  $F(t) = 1 - S(t)$  with respect to  $t$ . This provides  $f(t) = h(t) \exp\{-H(t)\}$ , where  $h(t)$  is given in Equation 1.

The quantile is an information of interest in many data analyses. In the context of the PE distribution, this quantity can be calculated via the relationship  $H(t) = -\ln(1 - F(t))$ . Given a probability  $p$ , the corresponding quantile is the value  $t^*$  such that  $H(t^*) = -\ln(1 - p)$ . Let  $w(p) = -\ln(1 - p)$  and recall that  $H(a_{m+1}) = +\infty$ . Then, the calculation of  $t^*$  can be expressed by:

$$t^* = \begin{cases} a_1 + \frac{w(p)}{\lambda_1}, & \text{if } w(p) \leq H(a_2) = \lambda_1; \\ a_j + \frac{w(p) - \sum_{i=1}^{j-1} \lambda_i(a_{i+1} - a_i)}{\lambda_j}, & \text{if } H(a_j) < w(p) \leq H(a_{j+1}), \quad j = 2, \dots, m. \end{cases} \quad (3)$$

Note that we can easily compute the median of the PE distribution by setting  $p = 0.5$  in Equation 3. Those readers interested in details about the first two moments of the PE distribution can refer to [Barbosa, Colosimo, and Neto \(1996\)](#).

### 3. The JAGS module

**JAGS** (“Just Another Gibbs Sampler”; see [Plummer 2003](#)) is a general-purpose, open-source, cross-platform graphical modeling program using a set of MCMC methods for stochastic simulations to generate samples from the joint posterior distribution of the parameters in a Bayesian model. These samples are further used for inferences regarding the target parameters and the model fit. A recent version of **JAGS** can be downloaded from <http://mcmc-jags.sourceforge.net/> and it is also available as a package for different Linux distributions.

A special feature of **JAGS** is that it can be modular and extensible with new functionalities, which can be loaded whenever required. A module can be created to accommodate all types of probability distributions (continuous or discrete, univariate or multivariate). In order to write these extensions, the knowledge of C++ and object oriented programming is necessary; however, we emphasize that installing the extensions can be extremely automatic depending on the operational system.

Technical manuals providing a comprehensive coverage on how to build a **JAGS** module were not available in the literature at the time of writing this material. We follow the steps indicated in [Wabersich and Vandekerckhove \(2014\)](#), which can be seen as a short tutorial exploring the key aspects to add custom distributions to **JAGS**. Those aspects not covered in this reference, but important to implement the PE distribution, are given careful attention in [Appendix A](#). For instance, the authors in [Wabersich and Vandekerckhove \(2014\)](#) work with two examples: the Bernoulli distribution for didactic purposes and the Wiener diffusion first-passage time distribution. In both cases the input values are scalar quantities; i.e., the paper does not explore distributions with vectors as arguments, which is the case for the  $PE(\lambda, \tau)$ . This issue can be easily addressed by simply modifying the parent class type expressed in the code from scalar to vector (see [Appendix A](#)).

The PE distribution implementation described in [Appendix A](#) is in fact a more complete example displaying the use of important resources available to create a **JAGS** module. Besides implementing the log-density and the sample generating process allowing the MCMC simulations in a Gibbs sampler, we have built auxiliary functions returning the pdf, cdf, quantile, hazard and cumulative hazard of the  $PE(\lambda, \tau)$  at a given time  $t$  or probability  $p$ . In addition, the module includes a routine to check whether the values in  $\lambda$  and  $\tau$  are consistent with the specifications shown in [Section 2](#). Hence, [Appendix A](#) contains essential complementary information (with respect to [Wabersich and Vandekerckhove 2014](#)) to those readers interested in extending **JAGS**.

### 3.1. Using the module in the R environment

In this section, we discuss how to use the proposed module in association with R. This is achieved through the package **rjags** ([Plummer 2019](#)) providing an interface between R and **JAGS**. Installing the new module is straightforward as indicated in [Appendix A](#). The procedure basically involves the installation of an R package called **pexm**. The package was designed for this task and also contains tools to help loading the module into the R environment. The **pexm** source can be downloaded from three repositories: CRAN, GitHub at <https://github.com/vdinizm/pexm> and SourceForge at <https://sourceforge.net/projects/jags-pexm/files> (see more details in [Appendix A.3](#)). Hereafter, we assume that a recent version of **JAGS**, R, **rjags** and the proposed **pexm** module are correctly installed in the computer. A confirmation that the module is ready to be used in R can be obtained by simply typing the following commands in the console:

```
R> library("pexm")
R> loadpexm()
```

In order to illustrate how to use the module, we have developed a simple example taking into account the R package **msm** ([Jackson 2011](#)) to generate the data. This package provides functions to fit continuous-time Markov and hidden Markov multi-state models and, in par-

ticular, includes the function `rpexp` to draw samples from the PE distribution. We can also use `dpexp`, `ppexp` and `qpexp` to explore the pdf, cdf and quantiles.

Let  $T_i$  be the survival time of the  $i$ -th individual,  $i = 1, \dots, n$ . In our illustration, we set the real rates  $\lambda = (0.3, 0.6, 0.8, 1.3)^\top$ , the time grid  $\tau = \{0, 2, 3, 5\}$ ,  $n = 1,000$  and assume  $T_i \sim \text{PE}(\lambda, \tau)$  to generate the data. The following objects are loaded into the R workspace: `t` is the observed vector  $(T_1, \dots, T_n)^\top$ , `n` is the length of `t`, `tau` is the grid vector, `m` is the length of `tau`, `pq` is a vector of probabilities for which the quantile should be evaluated, and `nq` is the length of `pq`.

### Code chunk CC.1:

```
R> data <- list(t = t, n = n, tau = tau, m = m, pq = pq, nq = nq)
R> parameters <- c("lambda", "ht100", "Ht100", paste0("q[", 1:nq, "]"),
+   paste0("loglik[", 1:n, "]"), paste0("St[", 1:n, "]"))
R> inits1 <- list(lambda = c(0.1, 0.5, 1, 2),
+   .RNG.name = "base::Super-Duper", .RNG.seed = 1)
R> inits2 <- list(lambda = c(0.5, 1.0, 1.5, 2.5),
+   .RNG.name = "base::Wichmann-Hill", .RNG.seed = 2)
R> burnin <- 1000
R> lag <- 1
R> npost <- 2000
R> Mjags <- rjags::jags.model(file = "Model_pex.R", data = data,
+   inits = list(inits1, inits2), n.chains = 2, n.adapt = burnin)
R> output <- rjags::coda.samples(Mjags, variable.names = parameters,
+   n.iter = npost, thin = lag)
```

The code chunk CC.1 shows a simple R script to organize the workspace information and then access **JAGS** for the MCMC simulation. Note that we specify: a list object (`data`) containing `t`, `n`, `tau`, `m`, `pq` and `nq`; a string vector (`parameter`) indicating which chains must be saved; two list objects (`inits1` and `inits2`) providing initial values for  $\lambda$  and setting details about random seeds in **JAGS**. The MCMC setup is determined through the elements: size of the burn-in period (`burnin`), thinning interval of the chains (`lag`) and the number of posterior samples required for inference (`npost`). The last two commands, `jags.model` and `coda.samples`, are functions from the library **rjags** to compile the Bayesian model in `Model_pex.R` (written in the **JAGS** syntax) and to run the corresponding MCMC algorithm, respectively. It is important to emphasize that the code chunk CC.1 assumes that the R working directory is the same one where the model script `Model_pex.R` is located. Note that we choose to handle the results in a format that can be conveniently analyzed via the R package **coda** (Plummer, Best, Cowles, and Vines 2006). Results are saved in the object `output`. The posterior estimates discussed ahead are based on the two chains generated for each parameter.

### Code chunk CC.2

```
model {
  for (i in 1:n) {
    t[i] ~ dpex(lambda[], tau[])
    St[i] <- 1 - ppex(t[i], lambda[], tau[])
    loglik[i] <- log(dpex(t[i], lambda[], tau[]))
  }
}
```

```

}
Ht100 <- hcpex(t[100], lambda[], tau[])
ht100 <- hpex(t[100], lambda[], tau[])
for (j in 1:m) {
  lambda[j] ~ dgamma(0.01, 0.01)
}
for (k in 1:nq) {
  q[k] <- qpex(pq[k], lambda[], tau[])
}
}

```

The content of the **JAGS** model script `Model_pex.R` is presented in the code chunk CC.2. This code indicates the likelihood  $T_i \sim \text{PE}(\lambda, \tau)$  and the prior specifications  $\lambda_j \sim \text{Ga}(0.01, 0.01)$  for  $j = 1, \dots, m$  (mean 1 and variance 100). Note that in each iteration of the MCMC, the survival function `St[i]` and the log-density `loglik[i]` are evaluated for each  $t_i$  together with the quantile `q[k]` for the probabilities in `pq`. In addition, the cumulative hazard function `Ht100` and the hazard function `ht100`, related to the time point  $t_{100}$ , are monitored in this example. As it can be seen, the elements defined in `Model_pex.R` explore all functionalities implemented for the PE distribution in the new module.

Note that, instead of saving the **JAGS** model in a separate file, one can also save the corresponding code as a string directly in R:

```

Model_pex <- "
model {
  for (i in 1:n) {
    t[i] ~ dpex(lambda[], tau[])
    St[i] <- 1 - ppex(t[i], lambda[], tau[])
    loglik[i] <- log(dpex(t[i], lambda[], tau[]))
  }
  Ht100 <- hcpex(t[100], lambda[], tau[])
  ht100 <- hpex(t[100], lambda[], tau[])
  for (j in 1:m) {
    lambda[j] ~ dgamma(0.01, 0.01)
  }
  for (k in 1:nq) {
    q[k] <- qpex(pq[k], lambda[], tau[])
  }
}
"
```

When calling the `jags.model` function, the command in code chunk CC.1 shall be modified as:

```

R> Mjags <- rjags::jags.model(textConnection(Model_pex), data = data,
+   inits = list(inits1, inits2), n.chains = 2, n.adapt = burnin)

```

### 3.2. Strategies to indirectly specify the likelihood

Without the proposed `pexm` module, one can use either the Poisson or the Bernoulli distribution to indirectly specify the likelihood function. These strategies are commonly known



as zeros-trick (Lunn *et al.* 2012) and ones-trick (Spiegelhalter *et al.* 2003), respectively. Let  $l_i = \ln[f(t_i|\lambda, \tau)]$ , then the likelihood can be written as:

$$f(t_1, \dots, t_n|\theta) = \prod_{i=1}^n \exp\{l_i\} = \prod_{i=1}^n \frac{\exp\{-(-l_i)\}(-l_i)^0}{0!} = \prod_{i=1}^n \text{P}(X_i = 0),$$

where  $X_i \sim \text{Pois}(-l_i)$ . In order to ensure a positive Poisson mean, we can select a constant  $C > 0$  and use:

$$\exp\{-C\} f(t_1, \dots, t_n|\theta) = \prod_{i=1}^n \frac{\exp\{-(-l_i + C)\}(-l_i + C)^0}{0!} = \prod_{i=1}^n \text{P}(X_i^* = 0),$$

where  $X_i^* \sim \text{Pois}(-l_i + C)$ . We must choose  $C$  such that  $-l_i + C > 0$  for all  $i = 1, \dots, n$ . Note that adding  $C$  does not affect the inference results, since it is equivalent to multiplying the posterior distribution by a constant term.

### Code chunk CC.3

```
model {
  C <- 10000
  for (i in 1:n) {
    zeros[i] ~ dpois( zero_mean[i] )
    zero_mean[i] <- -l[i] + C
    l[i] <- ... # log-likelihood here.
  }
  ... # Priors and other terms here.
}
```

This strategy can be applied in **JAGS** using the syntax shown in Code chunk CC.3. Note that the log-likelihood of the PE( $\lambda, \tau$ ) for the  $i$ -th observation must be specified for the object `l[i]`. This task for the PE context is not simple compared to other distributions. A possible approach is to replace the vector `t` and `tau` by other objects accounting for the position of each observed time point in the grid. These objects must be created in the R environment and passed to **JAGS** as a data argument (see `data` in the code chunk CC.1) together with `zero[i] = 0` for all  $i$ . They must allow the calculation of  $H(t)$  in **JAGS**, which in turn can be used to obtain  $\ln[f(t)] = \lambda_j - H(t)$  (if  $t \in I_j$ ) and  $S(t) = \exp\{-H(t)\}$ . After the MCMC simulation, the quantiles can be calculated in R using the posterior samples of  $\lambda$  and the function `qpexp` (package **msm**).

Again, we call attention to the fact that the Bernoulli distribution can also be used in a similar strategy expressing the model likelihood as  $\prod_{i=1}^n \text{P}(X_i^* = 1)$  with  $X_i^* \sim \text{Bern}(\exp\{l_i - C\})$ . Although both approaches have the same goal, the Poisson-zeros option is more popular in the literature.

The main conclusion of this section is that the **pexm** module provides a simpler and more attractive implementation of a Bayesian PE model in **JAGS**, since it avoids the use of alternative strategies to indirectly specify the likelihood. In the next section, we develop two types of studies using artificial data sets. The first one compares the results from the model in code chunk CC.2 and those obtained via the **msm** package. The second study is intended to evaluate the performance of **pexm** with respect to the Poisson-zeros strategy. In Section 4,



the analysis “**pexm** versus zeros-trick” is based on a real data set and it accounts for scripts (zeros-trick case) suggested in Ibrahim *et al.* (2001).

### 3.3. Simulation study

The first goal of the present section is to verify whether the proposed **JAGS** module is correctly implemented and working as expected. In order to carry out this type of analysis, we consider the configuration described in Section 3.1 to generate a sample of 1,000 time points from the PE distribution; the function **rpexp** (package **msm**) is applied here. In each interval defined by  $\tau$ , we have observed the following number of time points: 443 in  $I_1$ , 247 in  $I_2$ , 245 in  $I_3$  and 65 in  $I_4$ .

The R script shown in code chunk CC.1 is used to run the MCMC for the simple PE model indicated in code chunk CC.2. The argument **data** is set to evaluate **nq** = 23 different quantiles corresponding to probabilities in **pq** ranging from 0.01 to 0.99. A good behavior is confirmed, if all posterior estimates for  $\lambda$  are close to their corresponding real values (see Section 3.1) and the chain built for each log-density, survival function and quantile is centered around the true value calculated through one of the existing PE functions of the package **msm**. Figure 1 (a) clearly indicates this mentioned configuration. The three graphs show all true values (grey points) inside the 95% HPD (highest posterior density) intervals obtained based on the results from **JAGS**. In this paper, all HPD intervals are calculated through the R package **coda**. Other interesting aspects can be observed in Panel (a): (a) the shape and decay speed of the survival function; (b) the log density formed by four different parts corresponding to each interval  $I_j$ ; (c) the higher posterior uncertainty expressed in the log-density related to  $I_4$ . The behavior indicated in (c) is justified by the fact that only 65 time-points belong to the last grid interval, meaning less information to estimate  $\lambda_4$ .

Figure 1 (b) shows the chains generated for each  $\lambda_j$  after the burn-in period. As it can be seen, the convergence to the real values (horizontal lines) is visually confirmed and low autocorrelation is observed for all cases (recall that **lag** = 1). Here, all real values can be found within the corresponding 95% HPD intervals, and the posterior mean and median approximate well their targets. In terms of variability, the standard deviations can be ordered as follows:  $0.013 (\lambda_1) < 0.037 (\lambda_2) < 0.050 (\lambda_3) < 0.165 (\lambda_4)$ . Again, a larger number of observations in  $I_j$  leads to a smaller posterior variability for  $\lambda_j$ .

The code chunk CC.2 contains the **pexm** functions **hpex** and **hcpex**, which are alternatives to calculate during the MCMC run the hazard function in Equation 1 and the cumulative hazard function in Equation 2, respectively. Note that this **JAGS** model script is set to evaluate these functions for  $t_{100} = 3.483$ . Since  $\tau = \{0, 2, 3, 5\}$ , we have  $t_{100} \in I_3$  and thus  $h(t_{100}) = \lambda_3$ . The posterior sample related to the object **ht100** provides the mean 0.778 (the true  $\lambda_3$  is 0.8). The posterior mean of the cumulative hazard **Ht100** is 1.529 (the true value is 1.587). This average can also be obtained using the relationship  $H(t) = -\ln[S(t)]$ . A final remark related to these functions is that the present example can be extended to the survival regression setting. One might be interested in evaluating  $h(t)$  or  $H(t)$  under the effect of some fixed configuration of covariates. This can be easily adapted to **hpex** and **hcpex** by including the covariates effects in the argument **lambda**; Section 4 discusses two models having a regression setting.

#### Monte Carlo scheme

The second investigation developed in this section is based on a Monte Carlo (MC) scheme

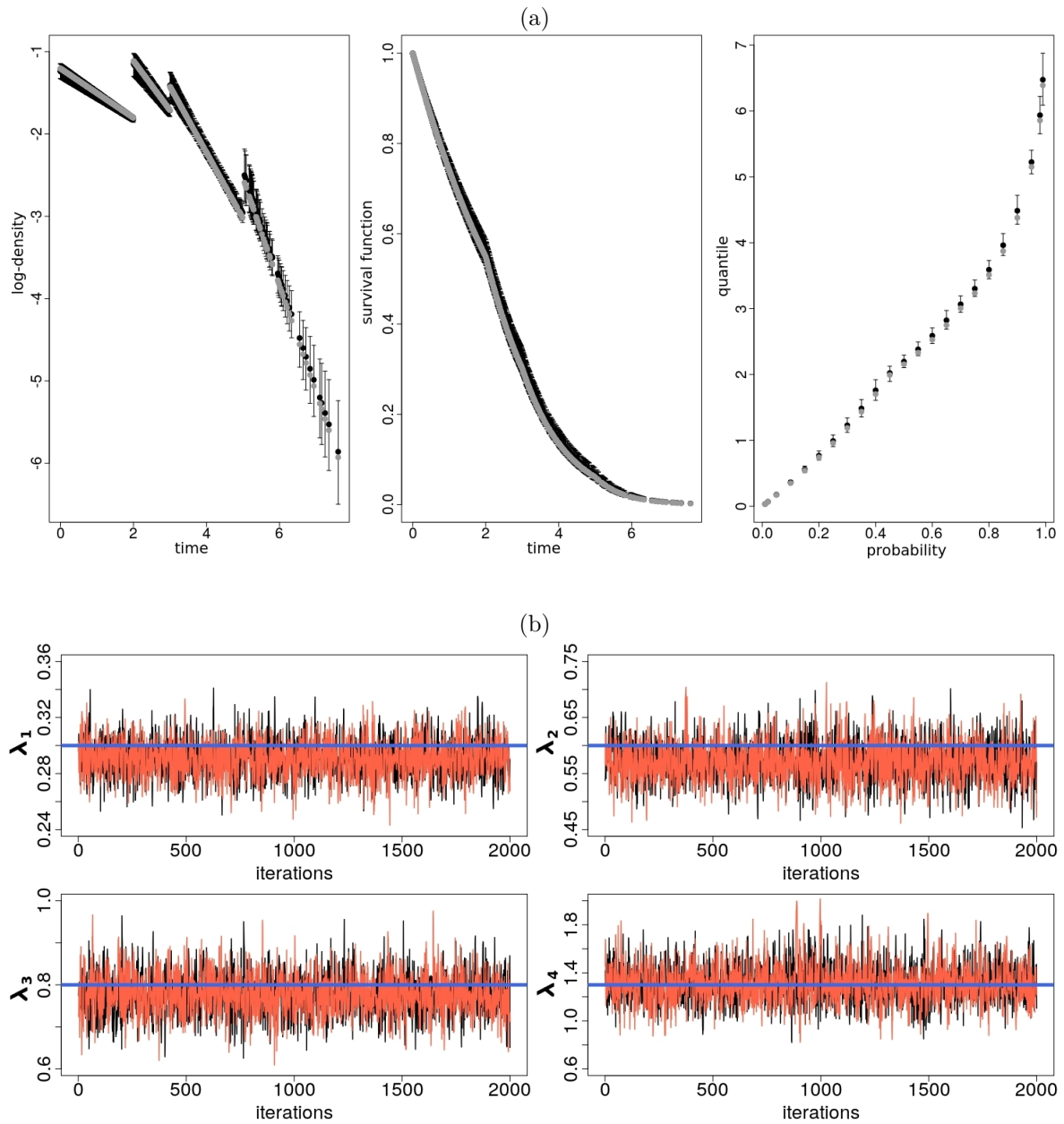


Figure 1: Comparison of posterior estimates (mean and 95% HPD interval) obtained from **JAGS** (black) and the values calculated via **msm** package (grey points). Panel (a): log-density and survival function for each observation  $t_i$  and quantiles for different probabilities. Panel (b): trace plots representing the two chains (in red and black after the burn-in period) for each  $\lambda_j$ ; horizontal lines (blue) indicate the real values.

with 100 replications, i.e., we evaluate 100 data sets generated under the same conditions. The main aim here is to compare the performance of a Bayesian model implemented with the module **pexm** and another version using the Poisson-zeros strategy (zeros-trick). We emphasize that the difference here is only the **JAGS** syntax. The model to be fitted is a simplification of the one exhibited in the code chunk CC.2. We will not save results related

to the quantiles, survival, hazard and cumulative hazard functions. In the **pexm** case, we only specify the likelihood via  $\mathbf{t}[i] \sim \text{dpex}(\text{lambda}[], \text{tau}[])$  and the prior  $\text{lambda}[j] \sim \text{dgamma}(0.01, 0.01)$ . On the other hand, the Poisson-zeros case is defined as indicated in code chunk CC.3, where the log-likelihood related to the PE distribution must be written for the object  $\mathbf{l}[i]$ ; we also set  $\lambda_j \sim \text{Ga}(0.01, 0.01)$ . Using this simple model, we evaluate “**pexm** versus zeros-trick” based on the MCMC outputs associated to the rates  $\lambda_j$ 's.

In order to generate the data sets, consider the steps: (a) set the grid  $\tau = \{0, 2, 3, 5\}$  establishing 4 intervals, (b) choose the sample size  $n$ , (c) define the true  $\lambda$  and (d) generate time observations from the PE distribution using the function **rpexp** from the R package **msm**. In the present analysis, we explore three scenarios of configurations for the true  $\lambda$ : increasing  $\lambda = \{0.3, 0.6, 0.8, 1.3\}$  (Scenario 1 or S1), constant  $\lambda = \{0.7, 0.7, 0.7, 0.7\}$  (Scenario 2 or S2) and decreasing  $\lambda = \{1.3, 0.8, 0.6, 0.3\}$  (Scenario 3 or S3). In addition to these scenarios, we evaluate two sample sizes ( $n = 100$  and  $n = 1,000$  time points).

For both implementations, the MCMC is configured with 1,000 burn-in iterations, lag equal to one and 2,000 posterior samples collected after the warm-up period. Two chains are obtained for each parameter. These chains are combined to calculate all descriptive statistics; except the effective sample size (using only the first chain). The starting values in the algorithm are also the same for both strategies; we highlight that the arguments **.RNG.name** and **.RNG.seed** (see the **rjags** documentation) were fixed to impose the same settings of random number generators in **JAGS**. The algorithms for all combinations of scenarios and sample sizes were executed in the same computer (Intel Core i7 processor and 16.00 GB memory) dedicated exclusively for this task. Figure 2 shows boxplots summarizing some of the most interesting results from the MC scheme. This type of graph is a good option in studies involving replications, since it indicates the variability of repeated measurements (Monte Carlo error) through the length of the boxes. Panel (a) presents the computational times (in seconds) needed to run the Gibbs sampling via **JAGS**. These values were determined through the existing function **proc.time** in R. The graphs in Panel (b) indicate the effective sample size of the chains obtained for the  $\lambda_j$ 's. This quantity was calculated using the function **effectiveSize** available in the R package **coda**. The light blue color represents the implementation based on the module **pexm**; light red is related to the Poisson-zeros strategy. Given that the same data set is being fitted by the same model with different implementations (this is done in a controlled setting for reproducibility) we have observed the exact same results from the corresponding MCMC runs. In other words, the outcomes of **pexm** and zeros-trick are the same in the comparison of bias, coverage percentages and autocorrelation. This aspect is clearly noted in Panel (b) and other statistics can be inspected through the supporting code in the replication materials of this paper.

In many studies, the computational time is measured in terms of the processor time used by each implementation to achieve a given effective sample size. This alternative cannot be applied here due to the mentioned compatibility of results. Instead, we focus on the required time to run 3,000 MCMC iterations for each artificial data set. Figure 2 (a) presents the main difference between **pexm** and the zeros-trick. Naturally, the computational times related to  $n = 100$  are smaller than those for fitting  $n = 1,000$ ; please note the distinct scales in the  $y$ -axis to allow visualization. For all combinations (scenario,  $n$ ), the boxplots associated with **pexm** are located in a lower level than those for the zeros-trick. This suggests that the computational speed is an advantage of the module.

Figure 2 also indicates that the differences between the scenarios S1, S2 and S3 are more

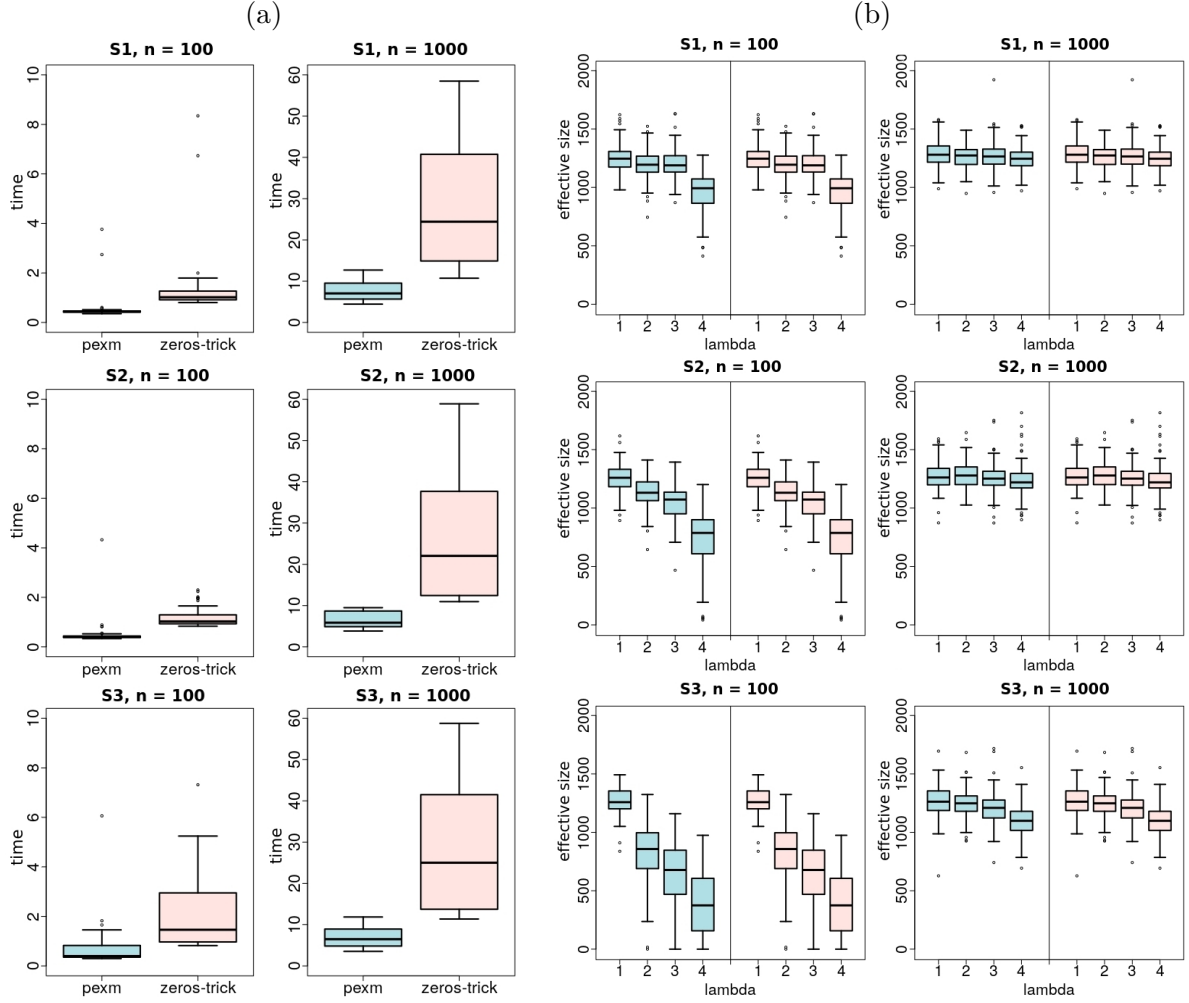


Figure 2: Boxplots summarizing the results from the MC scheme. Graphs in light blue (left) are related to the implementation via **pexm**. Graphs in light red (right) correspond to the Poisson-zeros strategy. Panel (a) shows the computational times required by **JAGS** to run the MCMC (Column 1 indicates  $n = 100$ , Column 2 indicates  $n = 1,000$ ). Panel (b) presents the effective sample sizes of the chains generated for each  $\lambda_j$  (Column 1 indicates  $n = 100$ , Column 2 indicates  $n = 1,000$ ). In both panels, the scenarios S1, S2 and S3 are represented in the rows 1, 2 and 3, respectively.

evident for the case  $n = 100$ . Note that the variability exhibited by the graphs are higher in (S3,  $n = 100$ ). Few distinctions are detected between the scenarios when  $n = 1,000$ . Panel (b) suggests that the effective sample size related to  $\lambda_1$  tend to be higher than those for the other rates. Recall, from the previous study, that the true  $\lambda$  in S1 provides few time points generated in the last interval of the grid ( $t_i > 5$ ). This feature also occurs for the choices of  $\lambda$  in S2 and S3. When comparing the number of cases  $t_i > 5$ , we have the following order  $S1 > S2 > S3$ . The small number of observations in the last interval (especially for S3) explains the worst performance of  $\lambda_4$  in terms of effective sample size.

## 4. Real application

The main goal of this section is to explore the `pexm` module in a Bayesian analysis via JAGS for a real data set. Two frailty PE models are considered here; they were proposed in Sahu *et al.* (1997) and reported in Ibrahim *et al.* (2001) for the kidney catheter data in McGilchrist and Aisbett (1991). The comparison “`pexm` versus zeros-trick” is also developed in this analysis.

The kidney catheter is a well known data set often used to illustrate survival models with random effects; it is also available in the R package `survival` (Therneau 2021).

```
R> data(kidney, package = "survival")
R> head(kidney)
```

	id	time	status	age	sex	disease	frail	
	1	1	8	1	28	1	Other	2.3
	2	1	16	1	28	1	Other	2.3
	3	2	23	1	48	2	GN	1.9
	4	2	13	0	48	2	GN	1.9
	5	3	22	1	32	1	Other	1.2
	6	3	28	1	32	1	Other	1.2

The response variable is the time to infection from the insertion of a catheter in a patient using portable dialysis equipment. The time of first and second infections are registered for each patient; there are 38 subjects and thus 76 time measurements for the analysis (18 of them are right-censored). After the occurrence (or censoring) of the first event, enough time is allowed to cure the infection before starting the second insertion. The data set includes an indicator variable identifying the event status (1 for infection, 0 for censoring) and three covariates: gender, age in years and disease type (with four categories). Following Ibrahim *et al.* (2001), the covariate “disease type” is not included in our study.

Let  $T_{ik}$  be the random variable representing the infection time for the  $i$ -th patient and the  $k$ -th insertion;  $i = 1, \dots, 38$  and  $k = 1, 2$ . In addition, consider the fixed covariate vector  $\mathbf{x}_{ik} = (x_{\text{sex } i}, x_{\text{age } ik})^\top$  where  $x_{\text{sex } i}$  is the gender and  $x_{\text{age } ik}$  is the age of the  $i$ -th patient in the  $k$ -th insertion. Given  $\mathbf{x}_{ik}$  and the unobserved positive random variable  $z_i$  (frailty), the conditional hazard function of  $T_{ik}$  in a shared frailty model can be written as:

$$h(t_{ik} | \mathbf{x}_{ik}, z_i) = h_0(t_{ik}) \exp\{\mathbf{x}_{ik}^\top \beta\} z_i, \quad (4)$$

with  $\beta = (\beta_{\text{sex}}, \beta_{\text{age}})^\top$  and  $h_0(\cdot)$  being the unknown baseline hazard function for all patients. The non-informative censoring mechanism is assumed in this application. Note that the random effect  $z_i$  establishes a dependency between the first and second infection times measured for the same patient. This modeling structure is common in the literature and it can be seen as an extension to the Cox proportional hazards model (Cox 1972). Frailty models were first motivated by Vaupel, Manton, and Stallard (1979) and then developed by Clayton and Cuzick (1985), Oakes (1986) and Oakes (1989).

The PE distribution, discussed in Section 2, can be used to represent the baseline hazard in Equation 4; that is  $h_0(t_{ik}) = \lambda_j$  for  $t_{ik} \in I_j$ ,  $j = 1, \dots, m$ . For simplicity, one can assume a prior configuration treating the  $\lambda_j$ 's as independent; however, Sahu *et al.* (1997) and

Aslanidou, Dey, and Sinha (1998) argue that in a frailty model the ratio of marginal hazards is similar to the ratio of baseline hazards for near time points (given the same covariates). In this case, it would be more appropriate to consider a prior that correlates the  $\lambda_j$ 's in adjacent intervals. Following Ibrahim *et al.* (2001), we shall explore two approaches:

- Model I:  $(\lambda_j | \lambda_{j-1}) \sim \text{Ga}(\alpha_j, \alpha_j / \lambda_{j-1})$  with  $\lambda_0 = 1$ ;
- Model II:  $\xi_j = \ln(\lambda_j)$  and  $(\xi_j | \xi_{j-1}) \sim N(\xi_{j-1}, \nu)$  with  $\xi_0 = 0$ .

In the structure of both models, we also have:  $z_i \sim \text{Ga}(\eta, \eta)$ ,  $\eta \sim \text{Ga}(\phi_1, \phi_2)$  and  $\beta \sim N_2[\mathbf{0}, \text{diag}(\sigma_1^2, \sigma_2^2)]$ . Denote  $\kappa = 1/\eta = \text{VAR}(z_i)$ ; large values of  $\kappa$  imply great heterogeneity between patients. We consider  $\phi_1 = \phi_2 = 10^{-3}$  and  $\sigma_1^2 = \sigma_2^2 = 10^3$ , which determines vague prior distributions (variance 1,000) for  $\eta$ ,  $\beta_{\text{sex}}$  and  $\beta_{\text{age}}$ . In addition, let  $\alpha_j = 10^{-2} \forall j$  in Model I and  $\nu = 10^4$  in Model II. The reader should refer to Ibrahim *et al.* (2001) for other comments related to these hyperparameters choices.

The authors in Ibrahim *et al.* (2001) also discuss a sensitivity analysis comparing  $m = 5$ , 10 and 20 intervals partitioning the time axis in the PE distribution. They indicate that  $m = 5$  provides the worst model fit, and the results for  $m = 20$  are not substantially better than  $m = 10$ ; therefore, for parsimony,  $m = 10$  is the choice selected for this analysis.

Due to the presence of right-censored observations, we use the standard strategy in **JAGS** based on the existing `dinterval` distribution to input the missing infection times; other details can be found in Appendix A. The **JAGS** scripts based on **pexm** for the models I and II are presented in the next three code chunks identified as CC.4 (a), (b) and (c). The code in (a) shows the main script containing: the time grid (line 1), the likelihood function (lines 3–12) and the prior distributions (line 13–18) of the frailties and the regression coefficients. Recall that the precision is the second parameter of the normal distribution, according to the **JAGS** parameterization. Note that the code in line 1 provides  $\tau = \{0, 56.2, 112.4, 168.6, 224.8, 281.0, 337.2, 393.4, 449.6, 505.8\}$ , which is an equally spaced time grid built with respect to the largest infection time (562) in the data set. The code chunks CC.4 (b) and CC.4 (c) show the commands related to the prior distributions correlating the  $\lambda_j$ 's in models I and II, respectively. Simply insert at the end of CC.4 (a) the script exhibited in (b) or (c) for the full model specification. The **JAGS** scripts for the PE models based on the zeros-trick are presented in Appendix B.

#### Code chunk CC.4 (a)

```
data {
  for(j in 1:m) { a[j] <- 562 * (j - 1) / m }
}
model{
  for (i in 1:n){
    for(k in 1:2){
      censored[i, k] ~ dinterval( t[i, k], t_cen[i, k])
      t[i, k] ~ dpex(haz[i, k, ], a[j])
      for(j in 1:m){
        haz[i, k, j] <- lambda[j] * exp( beta_sex * sex[i] +
                                          beta_age * age[i, k] ) * z[i]
      }
    }
  }
}
```



```

    }
  }
  for(i in 1:n){
    z[i] ~ dgamma(eta, eta)
  }
  kappa <- 1 / eta
  eta ~ dgamma(0.001, 0.001)
  beta_sex ~ dnorm(0, 0.001)
  beta_age ~ dnorm(0, 0.001)
  ... # Include here the Code chunk CC.4 (b) or (c).
}

```

#### Code chunk CC.4 (b)

```

rate[1] <- 0.01 / 1
lambda[1] ~ dgamma(0.01, rate[1])
for(j in 2:m){
  rate[j] <- 0.01 / lambda[j - 1]
  lambda[j] ~ dgamma(0.01, rate[j])
}

```

#### Code chunk CC.4 (c)

```

xi[1] ~ dnorm(0, 0.0001)
lambda[1] <- exp(xi[1])
for(j in 2:m){
  xi[j] ~ dnorm(xi[j - 1], 0.0001)
  lambda[j] <- exp(xi[j])
}

```

We can handle right-censored observations in **JAGS** by defining three variables to be inserted as data: (a) a binary indicator `isCensored` with 1 for censored values, (b) a time response vector (say `t_res`) with “NA” replacing the censored cases and (c) a censored time vector (say `t_cen`) having 0 for the non-censored subjects. The core of the **JAGS** model for censored data is represented by the two commands: `isCensored[i] ~ dinterval(t_res[i], t_cen[i])` and `t_res[i] ~ dpex(lambda[], tau[])`. Here, **JAGS** automatically imputes a random value for a missing observation `t_res[i] = NA`. The corresponding response `isCensored[i] = 1` determines that the observation to be imputed must be greater than `t_cen[i]`; this is implied by the existing **JAGS** distribution `dinterval` (see [Plummer 2003](#)). In this example, the new observation is then generated from the  $PE(\lambda, \tau)$  with lower bound `t_cen[i]` and unchanged upper bound  $+\infty$ .

The MCMC configuration for the present study is as follows: 10,000 burn-in iterations, lag equal to one, posterior sample composed by 10,000 observations and two chains being generated for each parameter. In terms of initial values, we set for both models the arguments: `.RNG.name = "base::Super-Duper"` and `.RNG.seed = 5` for the first chain and `.RNG.name = "base::Wichmann-Hill"` and `.RNG.seed = 2` for the second chain. See the **rjags** documentation ([Plummer 2019](#)) for further details about starting values in this tool.



Parameter	Model	Script	Mean	Median	SD	HPD
$\beta_{\text{sex}}$	I	Module	-1.4727	-1.4669	0.4888	(-2.3964, -0.4987)
		Poisson	-1.4764	-1.4639	0.4648	(-2.4195, -0.6104)
	II	Module	-1.4593	-1.4399	0.4675	(-2.3901, -0.5477)
		Poisson	-1.4650	-1.4529	0.4802	(-2.3769, -0.5111)
$\beta_{\text{age}}$	I	Module	0.0076	0.0071	0.0123	(-0.0165, 0.0319)
		Poisson	0.0055	0.0051	0.0115	(-0.0168, 0.0290)
	II	Module	0.0072	0.0070	0.0116	(-0.0161, 0.0295)
		Poisson	0.0063	0.0064	0.0123	(-0.0174, 0.0304)
$\kappa$	I	Module	0.5043	0.4642	0.2774	(0.0388, 1.0271)
		Poisson	0.4924	0.4507	0.2664	(0.0242, 0.9939)
	II	Module	0.4838	0.4430	0.2739	(0.0350, 0.9937)
		Poisson	0.4968	0.4502	0.2823	(0.0286, 1.0261)

Table 1: Posterior estimates and 95% HPD intervals for regression coefficients ( $\beta_{\text{sex}}$ ,  $\beta_{\text{age}}$ ) and variance ( $\kappa$ ) of the frailties. Comparisons: “Model I vs. II” and “**pexm** (or module script) vs. zeros-trick (or Poisson script)”.

Table 1 presents the posterior mean, median, standard deviation and 95% HPD credible intervals for  $\beta_{\text{sex}}$ ,  $\beta_{\text{age}}$  and  $\kappa$ . Here, we can compare the results from models I and II implemented with **pexm** and the Poisson-zeros strategy. In contrast with the simulation study, we do not observe here the exact same result when comparing the same model and parameter for both implementation types. However, the estimates obtained in this comparison are very similar; in some cases the distinction occurs in the third decimal place. This small difference can be justified by numerical approximation errors affecting the computations related to the more complex models under investigation in this section. Another interesting aspect to be highlighted is the similarity between the estimates from models I and II. As a result, one cannot choose one of the models based on a more reasonable interpretation of a parameter. Note that all credible intervals for  $\beta_{\text{age}}$  include zero, indicating that the age has no significant effect over the infection time. On the other hand, all intervals for  $\beta_{\text{sex}}$  are below zero, meaning that a female patient has lower infection risk than a male patient for a given time point.

According to Ibrahim *et al.* (2001), the posterior means of  $\kappa$  in Table 1 should be considered high, providing evidence of a strong positive association between two infection times for the same patient ( $T_{i1}$  and  $T_{i2}$ ). A large  $\kappa$  suggests that a major part of the variability in the data is explained by the clusters (the patients) rather than the insertions for the same individual.

The similarity of the estimates in Table 1, for the comparison “**pexm** versus zeros-trick”, suggests that the chains are converging to the same region in the parameter space and exhibiting equivalent variability. This behavior is expected since we fit the same model. The trace plots in Figure 3 confirm this interpretation and also indicate that the level of autocorrelation obtained via **pexm** tends to reflect the one produced via the Poisson-zeros strategy. The effective sample sizes of the chains, closely related to the autocorrelations, are shown at the top of the graphs. The values are not high, suggesting some strong autocorrelation that can be reduced by setting `lag` > 1 in the MCMC configuration. Although small ESS are reported, great discrepancies are not detected when comparing the implementations. The three horizontal

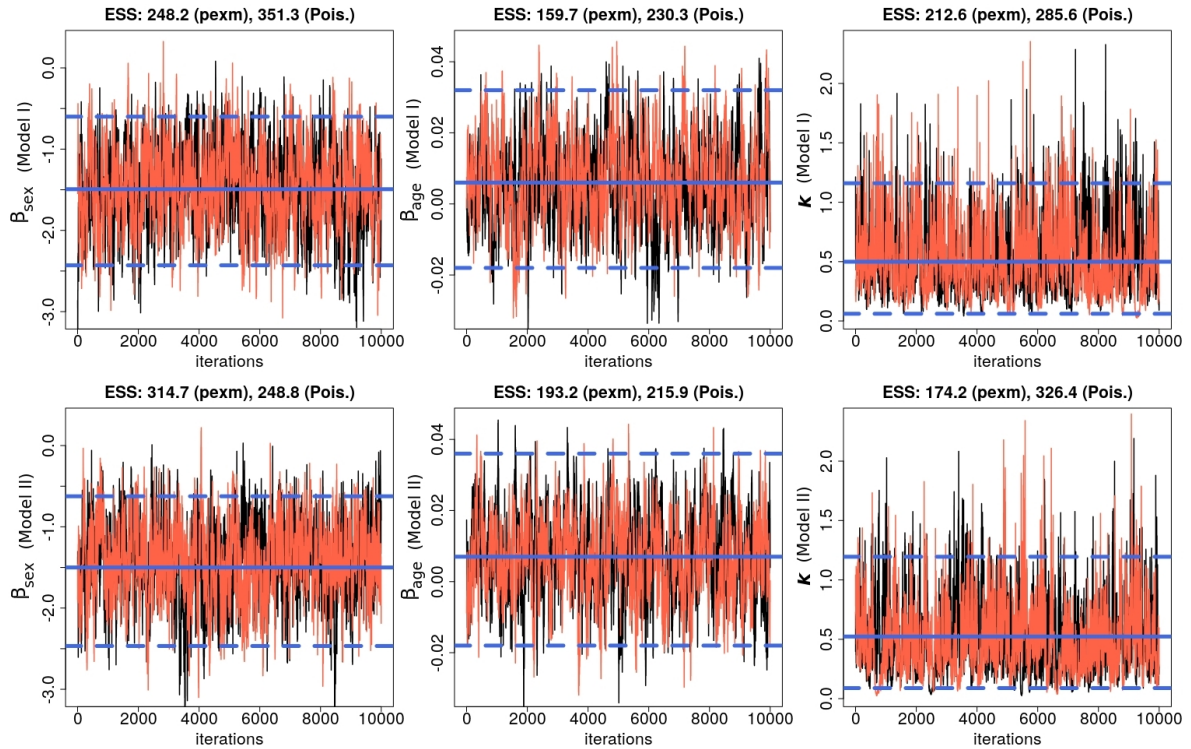


Figure 3: Trace plots representing the first chain (after burn-in period) for  $\beta$  and  $\kappa$ . The posterior samples related to **pexm** are in red and those related to the Poisson-zeros are in black. The horizontal lines indicate the estimates reported in Ibrahim *et al.* (2001): mean (continuous) and 95% credible intervals (dashed). Effective sample sizes (ESS) are presented at the top of each graph.

lines, included in each graph, represent the mean and the 95% credible interval reported in Ibrahim *et al.* (2001) for the corresponding parameter. Note that all chains are converging to the same region indicated in the literature.

Figure 4 shows the estimated kernel densities obtained in R for the posterior samples generated in this real application. Note that the curves in red (PE module) and black (Poisson-zeros) can be considered similar in all cases, which reinforces the idea of two algorithms working as expected to fit the same model. Very few discrepancies are detected in these graphs, especially around their modes; however, this can be significantly reduced when assuming a larger posterior sample size (say 100,000 iterations after the burn-in). In Figure 5, we compare the estimated baseline hazard functions  $h_0(t_{ik})$  obtained via **pexm** (red) and the zeros-trick (black). As it can be seen, the results from both cases tend to agree. A slightly difference is detected in the time interval related to  $\lambda_{10}$ . Such small distinction almost disappears when evaluating the posterior medians.

Table 2 presents the posterior standard deviations of each  $\lambda_j$ . Note that the variability tend to be higher when the index  $j$  is large. This aspect is related to the choice of the time grid  $\tau$ . Assuming  $m = 10$  and the equally spaced configuration, the number of observed time points (18 censored cases not included) in each interval  $I_j$  are: 30, 5, 9, 5, 1, 3, 0, 2, 0 and 3 for  $j = 1, \dots, 10$ , respectively. Therefore, the last intervals (and their neighbors) have fewer observations, which determines higher posterior uncertainty for the corresponding rates.

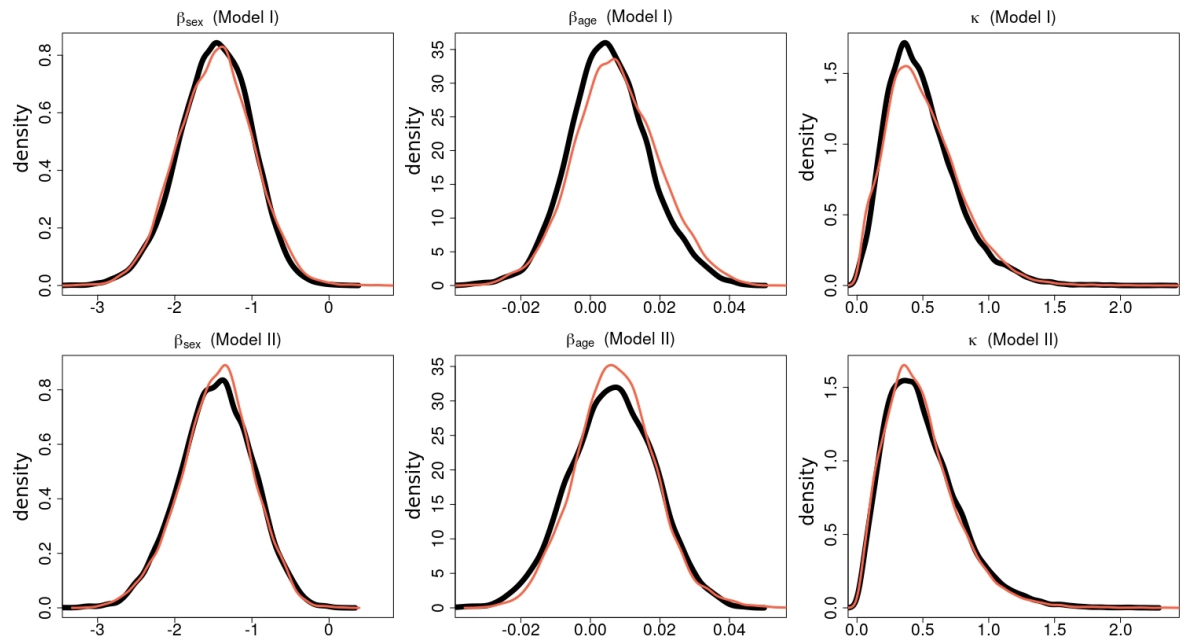


Figure 4: Kernel density estimation (assuming the default “Gaussian” smoothing kernel in R) for the posterior samples of  $\beta$  and  $\kappa$ . Results using the module are represented by the red thin line and the ones using the Poisson-zeros strategy in the black thick line.

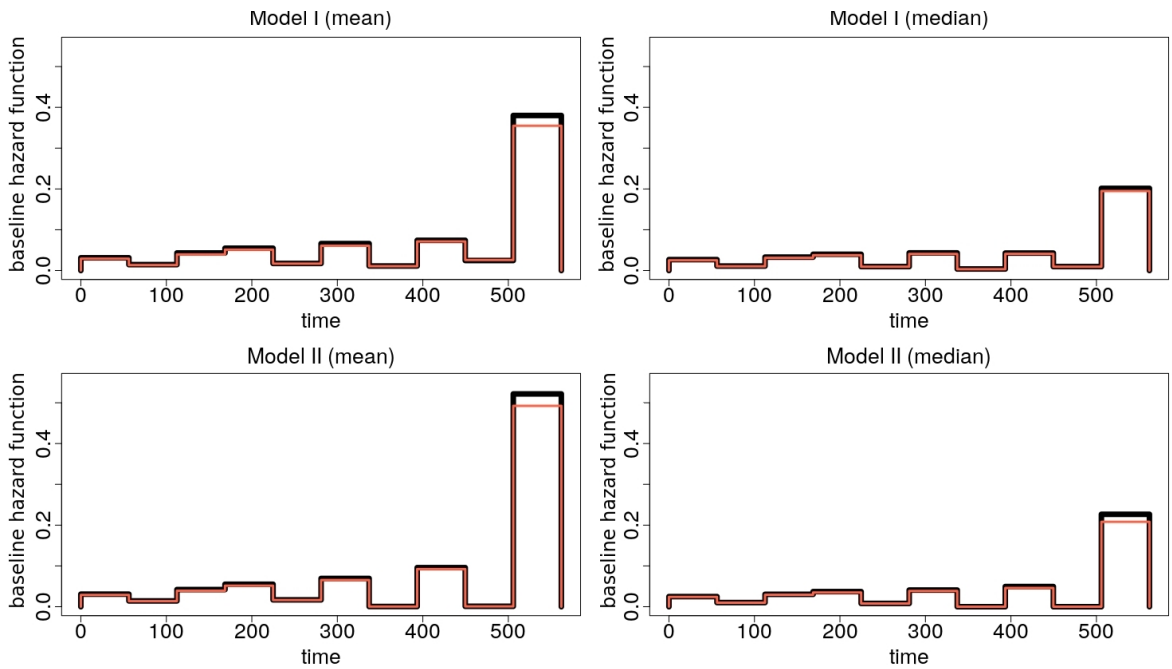


Figure 5: Comparison of posterior estimates for each  $\lambda_j$  indexing the PE distribution. Consider: *pexm* (red thin line), zero-trick (black thick line), mean (first column), median (second column), Model I (first row) and Model II (second row).

Model	Script	$\lambda_1$	$\lambda_2$	$\lambda_3$	$\lambda_4$	$\lambda_5$	$\lambda_6$	$\lambda_7$	$\lambda_8$	$\lambda_9$	$\lambda_{10}$
I	Module	0.0183	0.0109	0.0312	0.0481	0.0216	0.0623	0.0223	0.0916	0.0479	0.4570
	Poisson	0.0195	0.0122	0.0359	0.0503	0.0234	0.0740	0.0200	0.0910	0.0455	0.5568
II	Module	0.0194	0.0134	0.0351	0.0579	0.0296	0.1005	0.0043	0.1764	0.0122	1.1870
	Poisson	0.0222	0.0140	0.0401	0.0629	0.0310	0.0945	0.0058	0.1690	0.0208	1.2393

Table 2: Posterior standard deviations for each  $\lambda_j$ . Comparison: “Model I vs. II” and “**pexm** (or module script) vs. zeros-trick (or Poisson script)”.

The module **pexm** allows a simpler implementation of a Bayesian PE model, but this is not the only advantage observed in this paper. The MC study in Section 3.3 clearly indicates that the computational speed is another positive aspect of the proposed tool. The gain in terms of processor time can also be noted when running the MCMC related to the models I and II in the present real illustration. Using the same computer mentioned in the MC study (Section 3.3), the Poisson-zeros strategy is again considerably slower than the **pexm** implementation. The function `proc.time` in R provides the following times (in seconds) to execute the MCMC via **JAGS**: 24.493 (Model I, PE module), 23.954 (Model II, PE module), 70.857 (Model I, Poisson-zeros) and 68.168 (Model II, Poisson-zeros).

## 5. Conclusions

The PE distribution has been widely used in semiparametric settings to model data from applications in survival analysis and reliability. Implementing a PE model is a difficult task for anyone aiming a Bayesian analysis without the necessary programming skills to write all MCMC steps. In order to circumvent this issue, the programs based on the BUGS language are seen as attractive alternatives by many users. However, none of these programs has a module designed to deal with the PE distribution. In this case, a well known implementation strategy is to consider the Poisson or the Bernoulli distributions to indirectly specify the likelihood, which again poses some programming challenges.

**JAGS** deserves a special attention in the group of programs similar to BUGS, since it is free and extensible. The main goal of this paper was to present a new **JAGS** module (**pexm**) to fit Bayesian models accounting for the PE distribution. We have provided a clear picture of how the new module can be used from within the R environment (integrated with **JAGS**), which is important to guide the interested users. The discussion emphasizes that the module simplifies the **JAGS** model script.

A study based on artificial data was developed to compare results using **pexm** and those from an R package (**msm**) containing utilities for the PE distribution. The investigation concludes that the module works well. An MC simulation scheme with 100 replications, using a simple Bayesian model, was also conducted to evaluate the behavior of **pexm** with respect to the usual zeros-trick implementation. The main conclusion here is that the module provides the exact same results with the advantage of having higher computational speed.

In a real data application, we have considered a data set with two covariates and the response “time to infection” for patients using a portable dialysis equipment. Two PE survival frailty models in the literature were explored in this part. They assume different structures of association between adjacent intervals in the time grid. The presence of right censored data allowed a more in depth evaluation of the PE module; censoring was not considered in the

simulation study. Discussions were again devoted to the comparison “**pexm** versus zeros-trick”. As expected for two algorithms fitting the same model, their results were quite similar. The small estimation distinctions are possibly due to numerical approximation errors related to complexity of the tested models. In addition, the posterior estimates of the parameters resemble the ones reported in the literature using the same data set.

One last contribution of this work is the compilation of important information for those readers (with some C++ knowledge) interested in extending **JAGS**. The obstacles, solutions and other aspects related to the module implementation are fully described in Appendix A.

## Acknowledgments

The authors would like to thank two anonymous referees for their constructive comments leading to an improved version of this paper and to the R package developed to install the new module. The first author also thanks Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) for supporting this research.

## References

- Aslanidou H, Dey DK, Sinha D (1998). “Bayesian Analysis of Multivariate Survival Data Using Monte Carlo Methods.” *Canadian Journal of Statistics*, **26**, 33–48. doi:10.2307/3315671.
- Barbosa EP, Colosimo EA, Neto FL (1996). “Accelerated Life Tests Analyzed by a Piecewise Exponential Distribution via Generalized Linear Models.” *IEEE Transactions on Reliability*, **45**(4), 619–623. doi:10.1109/24.556584.
- Breslow N (1972). “Discussion on Regression Models and Life-Tables (by D. R. Cox).” *Journal of the Royal Statistical Society B*, **34**, 216–217.
- Breslow N (1974). “Covariance Analysis of Censored Survival Data.” *Biometrics*, **30**, 89–99. doi:10.2307/2529620.
- Brezger A, Kneib T, Lang S (2005). “**BayesX**: Analyzing Bayesian Structured Additive Regression Models.” *Journal of Statistical Software*, **14**(11), 1–22. doi:10.18637/jss.v014.i11.
- Carpenter B, Gelman A, Hoffman M, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). “**Stan**: A Probabilistic Programming Language.” *Journal of Statistical Software*, **76**(1), 1–32. doi:10.18637/jss.v076.i01.
- Clark DE, Ryan LM (2002). “Concurrent Prediction of Hospital Mortality and Length of Stay from Risk Factors on Admission.” *Health Services Research*, **37**, 631–645. doi:10.1111/1475-6773.00041.
- Clayton DG, Cuzick J (1985). “Multivariate Generalizations of the Proportional Hazards Model.” *Journal of the Royal Statistical Society A*, **148**, 82–117. doi:10.2307/2981943.
- Cox DR (1972). “Regression Models and Life-Tables.” *Journal of the Royal Statistical Society B*, **34**, 187–220. doi:10.1111/j.2517-6161.1972.tb00899.x.

- Demarqui FN, Dey DK, Loschi RH, Colosimo EA (2014). “Fully Semiparametric Bayesian Approach for Modeling Survival Data with Cure Fraction.” *Biometrical Journal*, **56**(2), 198–218. doi:10.1002/bimj.201200205.
- Denwood MJ (2016). “**runjags**: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in **JAGS**.” *Journal of Statistical Software*, **71**(9), 1–25. doi:10.18637/jss.v071.i09.
- Gamerman D (1991). “Dynamic Bayesian Models for Survival Data.” *Applied Statistics*, **40**(1), 63–79. doi:10.2307/2347905.
- Gamerman D (1994). “Bayes Estimation of the Piecewise Exponential Distribution.” *IEEE Transactions on Reliability*, **43**, 128–131. doi:10.1109/24.285126.
- Gamerman D, Lopes HF (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, volume 68. 2nd edition. Chapman & Hall/CRC, London. doi:10.1201/9781482296426.
- Gelfand AE, Smith AFM (1990). “Sampling-Based Approaches to Calculating Marginal Densities.” *Journal of the American Statistical Association*, **85**(410), 398–409. doi:10.1080/01621459.1990.10476213.
- Geman S, Geman D (1984). “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721–741. doi:10.1109/tpami.1984.4767596.
- Gilks WR (1992). “Derivative-Free Adaptive Rejection Sampling for Gibbs Sampling.” In JM Bernardo, JO Berger, AP Dawid, AFM Smith (eds.), *Bayesian Statistics 4*, pp. 641–649. Oxford University Press.
- Gilks WR, Best N, Tan K (1995). “Adaptive Rejection Metropolis Sampling within Gibbs Sampling.” *Journal of Royal Statistical Society C*, **44**, 455–472. doi:10.2307/2986138.
- Gilks WR, Thomas A, Spiegelhalter DJ (1994). “A Language and Program for Complex Bayesian Modelling.” *Journal of the Royal Statistical Society D*, **43**(1), 169–177. doi:10.2307/2348941.
- Gilks WR, Wild P (1992). “Adaptive Rejection Sampling for Gibbs Sampling.” *Journal of the Royal Statistical Society C*, **41**, 337–348. doi:10.2307/2347565.
- Hastings W (1970). “Monte Carlo Sampling Using Markov Chains and Their Applications.” *Biometrika*, **57**, 97–109. doi:10.1093/biomet/57.1.97.
- Hoffman MD, Gelman A (2014). “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.” *Journal of Machine Learning Research*, **15**, 1351–1381. URL <http://jmlr.org/papers/v15/hoffman14a.html>.
- Ibrahim JG, Chen MH, Sinha D (2001). *Bayesian Survival Analysis*. Springer series in statistics. Springer-Verlag, New York.
- Jackson CH (2011). “Multi-State Models for Panel Data: The **msm** Package for R.” *Journal of Statistical Software*, **38**(8), 1–29. doi:10.18637/jss.v038.i08.



- Kalbfleisch JD, Prentice RL (1973). “Marginal Likelihoods Based on Cox’s Regression and Life Model.” *Biometrika*, **60**, 267–278. doi:[10.1093/biomet/60.2.267](https://doi.org/10.1093/biomet/60.2.267).
- Kim JA, Proschan F (1991). “Piecewise Exponential Estimator of the Survivor Function.” *IEEE Transactions on Reliability*, **40**, 134–139. doi:[10.1109/24.87112](https://doi.org/10.1109/24.87112).
- Lunn D, Jackson C, Best NG, Thomas A, Spiegelhalter DJ (2012). *The BUGS Book: A Practical Introduction to Bayesian Analysis*. Chapman & Hall/CRC, Boca Raton.
- Mayrink VD, Duarte JDN, Demarqui FN (2021). *pexm: Loading a JAGS Module for the Piecewise Exponential Distribution*. R package version 1.1.1, URL <https://CRAN.R-project.org/package=pexm>.
- McGilchrist CA, Aisbett CW (1991). “Regression with Frailty in Survival Analysis.” *Biometrics*, **47**, 461–466. doi:[10.2307/2532138](https://doi.org/10.2307/2532138).
- Metropolis N, Rosenbluth A, Teller M, Teller E (1953). “Equations of State Calculations by Fast Computing Machines.” *Journal of Chemistry and Physics*, **21**, 1087–1091. doi:[10.2172/4390578](https://doi.org/10.2172/4390578).
- Neal R (2003). “Slice Sampling.” *The Annals of Statistics*, **31**, 705–767. doi:[10.1214/aos/1056562461](https://doi.org/10.1214/aos/1056562461).
- Oakes D (1986). “Semiparametric Inference in a Model for Association in Bivariate Survival Data.” *Biometrika*, **73**, 353–361. doi:[10.2307/2336211](https://doi.org/10.2307/2336211).
- Oakes D (1989). “Bivariate Survival Models Induced by Frailties.” *Journal of the American Statistical Association*, **84**, 487–493. doi:[10.1080/01621459.1989.10478795](https://doi.org/10.1080/01621459.1989.10478795).
- Plummer M (2003). “JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling.” URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.3406>.
- Plummer M (2010). *JAGS Developers Manual*. URL [https://sourceforge.net/projects/mcmc-jags/files/Manuals/2.x/jags\\_developer\\_manual.pdf](https://sourceforge.net/projects/mcmc-jags/files/Manuals/2.x/jags_developer_manual.pdf).
- Plummer M (2017). *JAGS Version 4.3.0 User Manual*. URL [https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/jags\\_user\\_manual.pdf](https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/jags_user_manual.pdf).
- Plummer M (2019). *rjags: Bayesian Graphical Models Using MCMC*. R package version 4-10, URL <https://CRAN.R-project.org/package=rjags>.
- Plummer M, Best N, Cowles K, Vines K (2006). “coda: Convergence Diagnosis and Output Analysis for MCMC.” *R News*, **6**(1), 7–11. URL <https://CRAN.R-project.org/doc/Rnews/>.
- R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Rigdon SE, Basu AP (2000). *Statistical Methods for the Reliability of Repairable Systems*. John Wiley & Sons, New York.



- Rue H, Martino S, Chopin N (2009). “Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations.” *Journal of the Royal Statistical Society B*, **71**(2), 319–392. doi:10.1111/j.1467-9868.2008.00700.x.
- Sahu SK, Dey DK, Aslanidou H, Sinha D (1997). “A Weibull Regression Model with Gamma Frailties for Multivariate Survival Data.” *Lifetime Data Analysis*, **3**, 123–137. doi:10.1023/a:1009605117713.
- Sinha D, Chen MH, Ghosh SK (1999). “Bayesian Analysis and Model Selection for Interval-Censored Survival Data.” *Biometrics*, **55**, 585–590. doi:10.1111/j.0006-341x.1999.00585.x.
- Sinha D, Dey DK (1997). “Semiparametric Bayesian Analysis of Survival Data.” *Journal of the American Statistical Association*, **92**(439), 1195–1212. doi:10.1080/01621459.1997.10474077.
- Spiegelhalter DJ, Thomas A, Best NG (2003). *WinBUGS Version 1.4 User Manual*. Cambridge: Medical Research Council Biostatistics Unit, URL <https://www.mrc-bsu.cam.ac.uk/wp-content/uploads/manual14.pdf>.
- Spiegelhalter DJ, Thomas A, Best NG, Lunn D (2014). *OpenBUGS User Manual*. Version 3.2.3, URL <http://www.openbugs.net/w/Manuals>.
- Stan Development Team (2020). “**rstan**: The R Interface to Stan.” R package version 2.21.2, URL <https://CRAN.R-project.org/package=rstan>.
- Stroustrup B (2013). *The C++ Programming Language*. 4th edition. Addison-Wesley.
- Therneau TM (2021). **survival**: *Survival Analysis*. R package version 3.2-11, URL <https://CRAN.R-project.org/package=survival>.
- Vaupel JM, Manton KG, Stallard E (1979). “The Impact of Heterogeneity in Individual Frailty on the Dynamics of Mortality.” *Demography*, **16**, 439–454. doi:10.2307/2061224.
- Wabersich D, Vandekerckhove J (2014). “Extending **JAGS**: A Tutorial on Adding Custom Distributions to **JAGS** (with a Diffusion Model Example).” *Behavior Research Methods*, **46**, 15–28. doi:10.3758/s13428-013-0369-3.
- Wickham H, Hester J, Chang W (2021). **devtools**: *Tools to Make Developing R Packages Easier*. R package version 2.4.2, URL <https://CRAN.R-project.org/package=devtools>.

## A. Technical aspects to build the proposed JAGS module

First, we highlight the fact that this description is based on the work of [Wabersich and Vandekerckhove \(2014\)](#). This reference is a nice short tutorial indicating the key steps to create a **JAGS** module. We call the attention for some important points required for the **pexm** implementation. The reader should refer to the module source code (under GPL-3 license) to follow the details reported here. The source code is available in three different repositories:

- <https://CRAN.R-project.org/package=pexm>,
- <https://github.com/vdinizm/pexm>,
- <https://sourceforge.net/projects/jags-pexm/files>.

Consider some generic notations related to important directories in the computer: `JAGS_PATH` is the path to the folder where **JAGS** is installed and `PEXM_PATH` is the path to the folder containing the source code of **pexm**. The **JAGS** modules are library files located in the `JAGS_PATH/module` directory of the program. Two C++ classes must be defined in the module code: one for the module itself and one for a distribution and their related functions (providing the pdf, cdf, quantiles, hazard and cumulative hazard). In our case, the module class file is called `pexm.cc` and it is located in the subfolder `PEXM_PATH/src` of the project. The first lines of this file references with `#include` the header files containing other functions to be used by the module. The first one is the existing **JAGS** header file `Module.h`. The second one is the PE distribution class file `DPex.h`. The remaining cases indicate function class files required to compute the density (`DPexFun.h`), cdf (`PPexFun.h`), quantile (`QPexFun.h`), hazard function (`HPexFun.h`) and cumulative hazard function (`HCPexFun.h`) given a time point  $t$  or a probability  $p$  of interest. These distribution and function class files are described ahead.

Note that we have chosen different names for the distribution class file (`DPex.h`) and the function class file to compute the pdf (`DPexFun.h`). This distinction is necessary in the module implementation, otherwise the system will indicate errors when creating the installation files. This is an important aspect to be clarified, because these class files are related to two **JAGS** model script commands having the same name but applied in different situations. As an illustration, consider:

- `T ~ dpex(lambda, tau)` is related to `DPex.h` and indicates that  $T \sim \text{PE}(\lambda, \tau)$ ;
- `Z <- dpex(t, lambda, tau)` is related to `DPexFun.h` and it allocates to the object `Z` the pdf of the  $\text{PE}(\lambda, \tau)$  evaluated at the time point  $t$ .

This equal name configuration is not mandatory, but it is in accordance with the naming standards adopted for the existing distributions available in **JAGS**.

### A.1. The distribution class files

The file `PEXM_PATH/src/distributions/DPex.h` contains the class prototype of the PE distribution. Near the top of the code, the **JAGS** parent class `VectorDist` is specified as the base class. This choice is appropriate for those distributions parameterized by vectors, which is the case of the  $\text{PE}(\lambda, \tau)$ . The examples in [Wabersich and Vandekerckhove \(2014\)](#) are associated with distributions taking scalar quantities as input arguments and thus related to

the alternative parent class `ScalarDist`. The process of building a **JAGS** module is easier for scalar valued distributions as the `RScalarDist` class can be inherited from, which partly automates the availability of `d/p/q` functions.

The following specific functions were implemented in the `pexm` C++ classes:

- `logDensity`: calculates the log-density;
- `randomSample`: generates random samples;
- `typicalValue`: returns a typical value in the support of the distribution;
- `checkParameterValue`: indicates if  $\{\lambda, \tau\}$  are in the allowed parameter space;
- `canBound`: indicates if the distribution can be bounded;
- `isDiscreteValued`: indicates if the PE distribution is discrete (false);
- `isSupportFixed`: indicates if the upper/lower limits of the PE distribution support are fixed (true, its support does not depend on  $\lambda$  or  $\tau$ );
- `checkParameterLength`: indicates if  $\lambda$  and  $\tau$  have the same length;
- `support`: returns the unbounded support  $(0, +\infty)$  of the PE distribution.

Some of these functions are not required when using the simpler `ScalarDist` parent class; see [Wabersich and Vandekerckhove \(2014\)](#) for this and other details related to the code.

The file `PEXM_PATH/src/distributions/DPex.cc` contains the actual implementation of the PE distribution. It includes the codes for the nine mentioned specific functions. In the beginning, note that we quote the name `dpex` and the number of parameters (i.e., 2) to be used in any **JAGS** script. The implementation of `logDensity`, `randomSample` and `typicalValue` takes into account the elements described in Section 2. In particular, `typicalValue` returns the median of the PE distribution, which is simpler to implement and uses the same structure presented ahead for the quantile function.

The function `randomSample` has two conditions related to the lower and upper bounds of the PE distribution. The default support is  $(0, +\infty)$ , however, this domain can be bounded to allow applications involving censored data (a very common situation in survival analysis and reliability). Given the existence of new boundaries, the commands implemented within `if(lower){...}` and `if(upper){...}` will replace the default interval  $(0, 1)$  by a shorter one, where a random number is uniformly generated and then transformed into a time point via cdf inversion. The specific function `canBound` is necessary to enable this restriction.

The function `checkParameterValue` allows **JAGS** to alert the user about the wrong specification of  $\lambda$  or  $\tau$ . The following points are verified: (a)  $\lambda_j \geq 0$  for all  $j$ , (b)  $a_1 = 0$  in  $\tau$ , (c)  $a_1 < a_2 < \dots < a_m$  in  $\tau$ . When running **JAGS**, a warning message will be displayed if at least one of these items is false.

## A.2. The function class files

The additional functions related to the  $PE(\lambda, \tau)$  are located in `PEXM_PATH/src/functions` within the project. The PE class prototype files associated with the pdf, cdf, quantile, hazard and cumulative hazard functions are denoted by `DPexFun.h`, `PPexFun.h`, `QPexFun.h`, `HPexFun.h` and `HCPexFun.h`, respectively. Their structures are quite similar, changing only the names from one case to the other. Near the top of each code, we set the **JAGS** function

base `ScalarVectorFunction.h` allowing the simultaneous specification of one scalar (time  $t$  or probability  $p$ ) and two vectors ( $\lambda$  and  $\tau$ ) as input arguments.

The directory `PEXM_PATH/src/functions` also contains the “.cc” files (they are `DPexFun.cc`, `PPexFun.cc`, `QPexFun.cc`, `HPexFun.cc` and `HCPexFun.cc`), where the actual implementations of the mentioned functions are found. The **JAGS** script names to call these functions (`dpex`, `ppex`, `qpex`, `hpex` and `hcpex`) and their number of input arguments (three) are defined at the beginning of each code. As an example, in a **JAGS** model script the user can allocate the corresponding outcome to the generic objects `Z1`, `Z2`, `Z3`, `Z4` and `Z5` as follows:

```
Z1 <- dpex(t, lambda, tau)
Z2 <- ppex(t, lambda, tau)
Z3 <- qpex(p, lambda, tau)
Z4 <- hpex(t, lambda, tau)
Z5 <- hcpex(t, lambda, tau)
```

Note that the first argument is always the scalar ( $t$  or  $p$ ), the second one is  $\lambda$  and the third one is the grid  $\tau$ .

### A.3. Installing the PE module

In order to configure the new **JAGS** module in the local computer, the user must simply install the R package **pexm** designed to handle this task in Unix-like (Linux or Mac) or Windows systems. See the repositories mentioned in Section A.1 to access the source code or the package tarball (`tar.gz` file) related to **pexm**. In the development of the proposed package, we considered as examples the source codes from **runjags** (Denwood 2016) and **rjags** (Plummer 2019), which contain key details about including modules to be used through the interface between R and **JAGS**. The **pexm** can be seen as an additional option in this group of examples to guide future developers. Code chunk CC.5 shows three possibilities to request the **pexm** installation in the R console: From CRAN with `install.packages()`, from GitHub with `install_github()`, or from SourceForge with `install.packages()`. We highlight two important notes here: (a) The package **devtools** (Wickham, Hester, and Chang 2021) is needed to deal with the option involving the GitHub repository. (b) The `tar.gz` file should be downloaded into the computer when using the SourceForge repository. The argument `"local_path_to_source"` in the code below needs to be replaced by the corresponding path to the `tar.gz` file saved in the computer. The reader must have in mind that a recent version of the program **JAGS** is expected to be installed in the system before applying any command listed in code chunk CC.5.

#### Code chunk CC.5

```
R> install.packages("pexm")
R> devtools::install_github("vadinizm/pexm")
R> install.packages("local_path_to_source", repos = NULL, type = "source")
```

Assuming that **pexm** was successfully installed in R, the user must first load the PE module before running the MCMC in **JAGS**. This task is done by typing `pexm::loadpexm()` in the R console. Further details about this function can be found in the **pexm** documentation.

## B. Additional information for the real application

The code chunk CC.6 shows the **JAGS** script for the zeros-trick implementation of the Bayesian models discussed in Section 4. The prior specification for the rates  $\lambda_j$ 's is supposed to be included in last line; consider the code chunk CC.4 (b) for Model I and CC.4 (c) for Model II. Line 2 specifies the time grid, the object `d[i, k, j]` indicates whether the event-time belongs to the interval  $I_j$  and the object `le[i, k, j]` receives the length of the intersection between the intervals  $(0, t_{ik})$  and  $I_j$ .

### Code chunk CC.6

```

data{
  for(j in 1:(m+1)) { a[j] <- 562 * (j - 1) / m }
  for(i in 1:n) {
    for(k in 1:2) {
      zeros[i, k] <- 0
    }
  }
}
model{
  for(i in 1:n){
    for(k in 1:2){
      for(j in 1:m){
        d[i, k, j] <- delta[i, k] * step(t[i, k] - a[j]) *
          step(a[j + 1] - t[i, k])
        le[i, k, j] <- (min(t[i, k], a[j + 1]) - a[j]) *
          step(t[i, k] - a[j])
        theta[i, k, j] <- lambda[j] * exp(beta_sex * sex[i] +
          beta_age * age[i,k] ) * z[i]
        mu[i, k, j] <- le[i, k, j] * theta[i, k, j]
        lik[i, k, j] <- dpois(d[i, k, j], mu[i, k, j])
      }
      loglik[i, k] <- sum(log(lik[i, k, ]))
      zeros[i, k] ~ dpois(-loglik[i, k])
    }
  }
  for(i in 1:n){ z[i] ~ dgamma(eta, eta) }
  kappa <- 1 / eta
  eta ~ dgamma(0.001, 0.001)
  beta_sex ~ dnorm(0, 0.001)
  beta_age ~ dnorm(0, 0.001)
  ... # Include here the Code chunk CC.4 (b) or (c).
}

```

**Affiliation:**

Vinícius D. Mayrink, João Daniel N. Duarte, Fábio N. Demarqui  
Departamento de Estatística  
Instituto de Ciências Exatas  
Universidade Federal de Minas Gerais  
Av. Antônio Carlos, 6627, Belo Horizonte, MG, Brazil, 31270-901  
E-mail: [vdm@est.ufmg.br](mailto:vdm@est.ufmg.br)