Flexible Regression Models for Count Data Based on Renewal Processes: The Countr Package

Tarak Kharrat
University of Liverpool

Georgi N. Boshnakov
University of Manchester

Ian McHale
University of Liverpool

Rose Baker
Salford Business School

Abstract

A new alternative to the standard Poisson regression model for count data is suggested. This new family of models is based on discrete distributions derived from renewal processes, i.e., distributions of the number of events by some time t. Unlike the Poisson model, these models have, in general, time-dependent hazard functions. Any survival distribution can be used to describe the inter-arrival times between events, which gives a rich class of count processes with great flexibility for modelling both underdispersed and overdispersed data. The R package **Countr** provides a function, `renewalCount()`, for fitting renewal count regression models and methods for working with the fitted models. The interface is designed to mimic the `glm()` interface and standard methods for model exploration, diagnosis and prediction are implemented. Package **Countr** implements state-of-the-art recently developed methods for fast computation of the count probabilities. The package functionalities are illustrated using several datasets.

This vignette is part of package **Countr**, version 3.5.3. For citations, please use the JSS version, Kharrat, Boshnakov, McHale, and Baker (2019) (see `citation("Countr")`).

Keywords: renewal process, duration dependence, count data, Weibull distribution, convolution, Richardson extrapolation.

1. Introduction

Modelling a count variable (the number of events occurring in a given time interval) is a common task in many fields such as econometrics, social sciences, sports modelling, marketing, physics or actuarial science just to name a few. The standard approach is to use the Poisson model, where $Y|x \sim \text{Poisson}(\lambda(x))$, where $\lambda(x) = \exp(x^\top \beta)$). Here Y is predicted given covariates with values x, using regression coefficients β. This model was built around a one-to-one correspondence between the count model (Poisson) and the distribution of the inter-arrival times (exponential). Perhaps this conceptual elegance contributed to its popularity. With this elegance comes some limitation: the Poisson model restricts the (conditional) variance to be equal to the (conditional) mean. This situation is rarely observed in real life data and among the thousands of alternatives proposed in the literature (see for example Winkelmann (2013) or Cameron and Trivedi (2013) for a review), only a few retain the correspondence between the

count model and the timing process. This correspondence is not only a conceptual elegance but also offers the researcher the flexibility to model the aspect (counting or timing) that is perhaps known better (from the available data) and to draw conclusions (typically prediction) using the other. A classic example is the exponential distribution used in radioactive decay which leads to Poisson count model. Another very good example in the marketing context can be found in McShane, Adrian, Bradlow, and Fader (2008).

Another limitation of the Poisson model results from the memorylessness property of the exponential distribution. This property states that the probability of having an arrival during the next $[t, t + \Delta t]$ time period (where $t > 0$ and $\Delta t > 0$) is independent of when the last arrival occured. In many situations, this assumption is not realistic and the history of the process can be informative about future occurrences.

One way to incorporate the history of the process in the modelling process is to make the current probability of an occurence depend on the number of previous event occurences. These models are known as *occurence dependence* and they are said to display true contagion. Bittner, Nussbaumer, Janke, and Weigel (2007) gave a discrete time example where the probability of scoring a goal in soccer in the current unit of time depends on the number of goals scored previously. The modelling process resulted in a negative binomial distribution.

Another way to take advantage of the process history is to assume that the time since the last observed event is informative about the probability of a future occurence. Inter-arrival times between events are still assumed to be independent and identically distributed but the hazard function, defined by $h(t) = f(t)/(1 - F(t))$, where $f(t)$ and $F(t)$ are the density and the cumulative probability function, is no longer a constant function of time (as in the exponential case) but is replaced by a time-varying function. These type of models display *duration dependence* where negative duration dependence is obtained by a decreasing hazard function (of time) and positive duration dependence by an increasing hazard function. As noted by Winkelmann (1995), *"Events are 'dependent' in the sense that the occurence of at least one event (in contrast to none) up to time t influences the occurence in $t + \Delta t$"*. This class of models is known as *renewal processes* and will form the main focus of this paper.

The key quantity when studying renewal processes (and time to event in general) is the hazard function. Not only does it fully characterize the inter-arrival timing distribution but it also relates to the type of dispersion observed in the corresponding count data. In particular, Winkelmann (1995) established that if the hazard function is monotonic, increasing (decreasing) hazard corresponds to count data with under-dispersion (over-dispersion); the constant hazard characterizing the exponential distribution corresponds to data with equi-dispersion. Therefore, allowing for a more flexible hazard function results in more flexible counting processes able to accomodate over-dispersed and under-dispersed, as well as equi-dispersed data.

Winkelmann (1995) was the first to comment on the usefulness of renewal process models and derived a count model based on gamma distributed inter-arrival times. The choice of the gamma distribution was justified by computational necessity. In fact, the reproductive property of the gamma distribution (sums of independent gamma random variables are gamma distributed) leads to a simple form for the derived gamma count probability. McShane *et al.* (2008) derived a closed formula for the count probability of a renewal process based on Weibull inter-arrival times using series expansion. The same approach has been used by Jose and Abraham (2011) and Jose and Abraham (2013) to derive a counting process with Mittag-

Leffler and Gumbel inter-arrival times, respectively.

Despite the attractive properties of count models based on renewal processes, their use is still limited in practice where Poisson, geometric and negative binomial are usually preferred. Perhaps the main reason is the lack of available software to easily fit this new type of models. The development of **Countr** (Kharrat and Boshnakov 2016), available from the Comprehensive R Archive Network (CRAN), is meant to fill in this gap and complete the practioners' toolbox for modeling count data in R (R Core Team 2017).

The **Countr** package provides a function, `renewalCount()`, for fitting count regression models based on renewal distributions. It offers several built-in inter-arrival times distributions and supports custom distributions. The design of the fitting function (`renewalCount()`) and the methods that act on the object returned by it, is meant to mimic the familiar user interface associated with a number of R modelling functions, especially `glm()` (Chambers and Hastie 1992) from package **stats** (R Core Team 2017), `hurdle()` and `zeroinfl()` (Zeileis, Kleiber, and Jackman 2008) and `flexsurvreg()` (Jackson 2016).

The remainder of this paper is laid out as follows. In Section 2, we briefly review the fundamental relationship between a timing process and the resulting count model, the different computation methods as well as the renewal regression models considered in **Countr**. The package design is discussed in Section 3 and a first working example is given in Section 4. A second extended example is analysed in Section 5 and is used to discuss the package functionality. A strategy to discriminate between models is suggested in Section 6 and illustrated with a real dataset. We conclude and discuss future work in Section 7.

2. Models

2.1. Count models and inter-arrival times

The distribution of non-negative integer valued discrete random variables, count distributions for short, can be used as the distribution of the number of events in a given time interval, and vice versa. A powerful method to specify count distributions then can be based on models of the times between the events.

Consider a stochastic process starting at time $t = 0$ which produces a sequence of events. Let τ_1 be the time of the first event and, in general, τ_k be the time between the $(k-1)$th and the kth event, $k \in \mathbb{N}$. The τ_k's are known as *inter-arrival times* or *waiting times*. The arrival time of the mth event is

$$a_m = \sum_{k=1}^{m} \tau_k, \qquad m = 1, 2, \ldots,$$

with cumulative probability function $F_m(t) = \mathsf{P}(a_m < t)$.

Let $N_t = N(t)$ denote the total number of events in $[0, t)$. For any fixed t (the observation horizon), N_t is the count variable we wish to model. We have $\mathsf{P}(N_t \geq m) = F_m(t)$ and $\mathsf{P}(N_t < m) = 1 - F_m(t)$, since $N_t \geq m$ if and only if the mth event occurs before time t.

Moreover, the probability, $P_m(t)$, for exactly m events before time t is

$$\begin{aligned} P_m(t) &\equiv \mathsf{P}(N_t = m) \\ &= \mathsf{P}(N_t \geq m) - \mathsf{P}(N_t \geq m+1) \\ &= F_m(t) - F_{m+1}(t) \end{aligned} \tag{1}$$

For fixed t, Equation (1) shows how a count distribution, $\{P_m(t),\ m = 0, 1, \ldots\}$, can be obtained from $\{F_m(t),\ m = 0, 1, \ldots\}$, which in turn can be specified flexibly by the inter-arrival distributions.

More specifically, let $\{\tau_k\}_{k \in N}$ be independent and identically distributed (iid) random variables with common density $f(\tau)$. In this case the process is called a *renewal process* (see Feller 1971, for a formal definition) and Equation (1) can be used to derive the following recursive relationship:

$$\begin{aligned} P_{m+1}(t) &= \int_0^t F_m(t-u)\,\mathrm{d}F(u) - \int_0^t F_{m+1}(t-u)\,\mathrm{d}F(u) \\ &= \int_0^t P_m(t-u)\,\mathrm{d}F(u), \qquad \text{for } m = 1, 2, \ldots, \end{aligned} \tag{2}$$

where $P_0(u) = S(u) = 1 - F(u)$ (a survival function). Equation (2) can be understood intuitively: the probability of exactly $m+1$ events occurring by time t is the probability that the first event occurs at time $0 \leq u < t$, and that exactly m events occur in the remaining time interval, integrated over all times u. $P_1(t), \ldots, P_m(t)$ can be generated in turn by evaluating this integral.

2.2. Count probability computation methods

Convolution methods

To compute the integral defined in Equation (2), one can adapt composite midpoint rule (e.g., Press, Teukolsky, Vetterling, and Flannery (2007, section 4.1.4)):

$$\int_0^{Nh} f(x)\,\mathrm{d}x = h \sum_{j=1}^{N} f\{(j - 1/2)h\} + O(h^2), \tag{3}$$

where there are N steps with stepsize h, and $Nh = t$. Note that the integrand i.e., f is not evaluated at the limits of the integral (open rule). Furthermore, if we define $g(u) = P_m(t-u)$ for some values of the count m and the time t (fixed) and $F(t)$ the CDF of the inter-arrival times distribution, the previous integral can be seen as the sum of N integrals of the form:

$$\int_{(j-1)h}^{jh} g(u)\,\mathrm{d}F(u) \simeq g\{(j - 1/2)h\}(F\{jh\} - F\{(j-1)h\}),$$

In order to reach $P_m(t)$, the previous computation requires all the previous m probabilities to be available. The algorithm starts by initialising a (local) q array to contain the P_0 at the midpoints $h/2 \cdots (N - 1/2)h$, sets up another local array to contain $F\{jh\} - F\{(j-1)h\}$, and carries out the convolutions. At the end of this step, the array $q[\,]$, initially containing

P_0, will be overwritten to contain P_1. These steps are repeated until the desired probability is obtained. This method was named the *direct* method in **Countr** and it has been shown in Baker and Kharrat (2017, Section 3) to have a $O(mN^2)$ complexity.

The *direct* method computes all probabilities up to the mth, which is slow if we need only the mth probability. It can be improved so that computing time is $O(\ln(m)N^2)$ instead of $O(mN^2)$, using the addition chain method (an adaptation of the method used by compilers for fast computation of integer powers of a variable with the minimum number of multiplications). We label this method *naive* method in **Countr** and refer readers to Baker and Kharrat (2017, Appendix A) for computation details.

A more efficient method to directly compute the mth probability is based on De Pril (1985) algorithm (and hence is called the *dePril* method in **Countr**). It has been shown to have a $O(N^2)$ complexity (Baker and Kharrat 2017, Section 4) and hence is the recommended (and the default) method in **Countr**. Readers interested in the computation details are referred to Depril's paper and Baker and Kharrat (2017, Section 4). Here we simply describe the main idea of the algorithm. Let q_i be the value of probability density function of the survival distribution evaluated at points $t_i \geq 0$ where $q_0 > 0$. Then the probability of m events is $f_N^{(m)}$, the m-fold convolution of q, given by

$$f_0^{(m)} = q_0^m,$$

and for $N > 0$ by the recursion

$$f_N^{(m)} = q_0^{-1} \sum_{j=1}^{N} \left(\frac{(m+1)j}{N} - 1\right) f_{N-j}^{(m)} q_j. \tag{4}$$

This algorithm when applied to our case requires three arrays: one to hold the survival function, one for the probability mass q, and one work array to hold f. To apply this method to continuous distributions like the Weibull, we first discretised the distribution, so that $q_j = F((j+1)h) - F(jh)$.

Improvement by Richardson extrapolation

The integration rule described in Equation (3) generates approximations of order $O(h^2)$. Richardson extrapolation can be used to progressively remove errors of order h^2, h^4 etc. Clearly, if an estimate $S_1 = S + \gamma h^\delta$ and $S_2 = S + \gamma(h/2)^\delta$, where S_1 and S_2 are the approximations with N and $2N$ steps respectively and S is the true value, we can remove the error and estimate S as

$$S_3 = (2^\delta S_2 - S_1)/(2^\delta - 1). \tag{5}$$

Subsequently, higher-order errors can be removed in the same way until the required accuracy is attained. Romberg integration can also be done with the extended-midpoint rule (e.g., Press *et al.* (2007)). The situation for convolutions is less straightforward, but a satisfactory solution was derived in Baker and Kharrat (2017, Appendix B). Clearly, the Richardson extrapolation has the appealing property of improving the accuracy, without necessitating a large value of N and consequent slow computation. Therefore, for all the built-in distributions, the default behaviour in **Countr** is to apply the Richardson extrapolation. Whenever it is possible, users are advised to activate the extrapolation option.

The special case of the Weibull distribution

In **Countr**, methods inspired from McShane *et al.* (2008) have been implemented when the inter-arrival times are Weibull distributed. In this case, the exponential in the Weibull density can be expanded out and series transformation techniques can be used to speed up convergence. Two algorithms are available: a matrix approach using a specified number of terms and a series accelerated method based on the Euler and van-Wijngaarden transformations (Press *et al.* 2007, Chapter 5) controlled by a number of iterations and a convergence tolerance parameters.

Naming conventions

We use the term *count distribution* or *renewal count distribution* for the distribution of N_t and qualify it with the name of the inter-arrival distribution for a particular distribution of the inter-arrival times. For example, *Weibull count distribution* refers to the count model arising from a renewal process with inter-arrival times having a Weibull distribution.

2.3. Renewal regression models

The regression models fitted by **Countr** are in the spirit of the generalised linear models (McCullagh and Nelder 1989) and consist of two main components: a conditional distribution of the response variable (given the covariates, if any) and one or more linear equations relating parameters to covariates, possibly via link functions.

More formally, let $\boldsymbol{Y}$ be the response variable of interest, $\boldsymbol{x}$ a vector of covariates and $\mathcal{D}$ a renewal count distribution. We assume that

$$\boldsymbol{Y}|\boldsymbol{x} \sim \mathcal{D}(\boldsymbol{\theta}), \tag{6}$$

where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)^\top$ is the vector of the parameters of $\mathcal{D}$.

One or more parameters of the distribution may depend linearly on covariates via link functions. The equation for the kth parameter then is:

$$g_k(\theta_k) = \boldsymbol{x}^\top \boldsymbol{\beta}_k, \tag{7}$$

where g_k is the link function for the kth parameter, $\boldsymbol{x}$ the covariates and $\boldsymbol{\beta}_k$ the corresponding vector of regression parameters. Typically, covariates are related to a location parameter but it is helpful in some applications to be able to let other parameters depend on covariates.

We call these models *renewal regression models*. Note that, in general, the renewal distributions are not from the exponential family. For comparison, in standard generalised linear models (GLM) the distribution is taken from the exponential family of distributions and the mean, transformed by a link function, is a linear combination of the covariates.

The inter-arrival distribution

Table 1 gives count distributions available in **Countr** and the corresponding inter-arrival time distributions from which they are obtained.

The Poisson distribution is the only one with a simple closed form expression. The other distributions provide alternatives, which extend the range of data that can be modelled with

Count distribution	$P(Y=k)$	Inter-arrival distribution	pdf $f(t)$	Parameters
Poisson	$\frac{\lambda^k}{k!}\exp^{-\lambda}$	Exponential	$\lambda e^{-\lambda t}$	λ
Weibull-count	NSCF	Weibull	$\lambda\beta t^{\beta-1}e^{-\lambda t^\beta}$	λ (scale), β (e^{shape})
Gamma-count	NSCF	Gamma	$\lambda^k t^{k-1}e^{-\lambda t}/\Gamma(k)$	λ (rate), k (shape)
Gengamma-count	NSCF	Gen. gamma	see Equation (8)	μ, σ, q
Burr-count	NSCF	Burr	$\frac{kc(t/\alpha)^{c-1}}{\alpha\left(1+(t/\alpha)^c\right)^{k+1}}$	α (scale), c (shape1), k (shape2)

Table 1: Built-in count distributions in package **Countr** and the interarrival time distributions generating them. NSCF stands for *no simple closed form*. $\Gamma(k)$ is the gamma function and the Burr-count uses the Burr type XII parameterization (Tadikamalla 1980).

count regression models. For example, they can accommodate over- and under-dispersion. Also, the systematic way in which these count distributions are derived may give advantageous insight in some cases.

It is also noteworthy that both the Weibull and gamma count models nest the basic Poisson model. In fact, setting $\beta = 1$ in the Weibull case or $k = 1$ in the Gamma case leads to the exponential distribution. Another interesting distribution that could be used with the convolution method is the generalised gamma first introduced by Stacy (1962). Prentice (1974) proposed an alternative parametrization which is preferred for computation. In the Prentice (1974) parametrization, the distribution has three parameters (μ, σ, q), and its survival function is given by:

$$S(t) = \begin{cases} 1 - I(\gamma, u) & \text{if q} > 0 \\ 1 - \Phi(z) & \text{if q} = 0 \end{cases} \tag{8}$$

where $I(\gamma, u) = \int_0^u x^{\gamma-1}\exp(-x)/\Gamma(\gamma)$ is the regularised incomplete gamma function (the gamma distribution function with shape γ and scale 1), Φ is the standard normal distribution function, $u = \gamma\exp(|q|z), z = (\log(t) - \mu)/\sigma$, and $\gamma = 1/q^2$. This distribution includes the Weibull (when $q = 1$), gamma (when $q = \sigma$) and log-normal (when $q = 0$) as special cases.

The default links (the functions $g_k()$ in equation (7)) associated with the built-in distributions are given in Table 2.

Count distribution	dist	Par. 1	Link	Par. 2	Link	Par. 3	Link
Weibull-count	`Weibull`	`scale`	log	`shape`	log		
Gamma-count	`gamma`	`rate`	log	`shape`	log		
Gengamma-count	`gengamma`	`mu`	log	`sigma`	log	`Q`	I()
Burr-count	`Burr`	`scale`	log	`shape1`	log	`shape2`	log

Table 2: Parameters and default link functions for the built-in count distributions in package **Countr**. I() stands for the identity function.

As discussed before, count models arising from renewal processes provide very flexible families of distributions. Perhaps the simplest way to use them is to simply ignore their connections to renewal theory. Several models can be tried and users can discriminate between models

using the following strategy:

- when models are nested, a likelihood ratio test (LR) statistic can be used. This is possible because renewal-count models are fully parametric and in this case the LR statistic has the usual $\chi^2(p)$ distribution, where p is the difference in the number of parameters in the model.
- when models are not nested, one can compare information criteria such as the Akaike information criterion (AIC) or the Bayesian information criterion (BIC) to choose the model that provides the best fit to the data.

This strategy is illustrated in Section 6.

In some applications however, the researcher may have some information about the inter-arrival time process which can lead to a particular choice of model. For example, assume that a researcher is interested in modelling the number of occurences by some time horizon t. He has data on the observed count for a number, n, of individuals, together with a set of individual covariates $\boldsymbol{x}_i, i = 1, \ldots, n$. If data on time to first event are also available, the researcher can fit a parametric hazard model using package **flexsurv** (Jackson 2016), choose the parametric model that presents the best fit and use the associated renewal count family to model his data. This approach has been used in Kharrat (2016, Chapter 4).

Parameters estimation

Parameter estimation is performed by maximum likelihood (ML). Define the log-likelihood $\mathcal{L} = \sum_{i=1}^{n} lnP_{y_i}(t|\boldsymbol{x}_i, \boldsymbol{\beta}_i)$, where $\boldsymbol{\beta}$ is the vector of parameters. The ML estimator $\hat{\boldsymbol{\beta}}$ is the solution of the first-order conditions,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \frac{\partial lnP_i}{\partial \boldsymbol{\beta}} = 0, \tag{9}$$

where $P_i = P_{y_i}(t|\mathbf{x}_i, \boldsymbol{\beta}_i)$ and $\partial \mathcal{L}/\partial \boldsymbol{\beta}$ is a $q \times 1$ vector.

Let $\boldsymbol{\beta}_0$ be the *true* value of $\boldsymbol{\beta}$. Using ML theory, we obtain $\hat{\boldsymbol{\beta}} \xrightarrow{p} \boldsymbol{\beta}_0$ and

$$\sqrt{n}(\hat{\boldsymbol{\beta}}_{ML} - \boldsymbol{\beta}_0) \xrightarrow{d} \mathcal{N}[\mathbf{0}, \mathbf{V}^{-1}], \tag{10}$$

where the $q \times q$ matrix $\mathbf{V}$ matrix is defined as

$$\mathbf{V} = -\lim_{n\to\infty} \frac{1}{n} \mathsf{E}\left[\sum_{i=1}^{n} \frac{\partial^2 \ln P_i}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}'}|_{\boldsymbol{\beta}_0}\right]. \tag{11}$$

To use this result, we need a consistent estimator of the variance matrix $\mathbf{V}$. Many options are available: the one implemented in **Countr** is known as the *Hessian estimator* and simply evaluates Equation (11) at $\hat{\boldsymbol{\beta}}$ without taking expectation and limit.

Goodness-of-fit

For fully parametric models such as Poisson or renewal-count, a crude diagnosis is to compare the fitted probabilities with observed frequencies. Things are better understood with a

formula. Define the count variable $y_i, i = 1, \ldots, n$, where n is the total number of individuals and let $m = max(y_i)$. We denote by $\bar{p}_j$ the observed frequencies (the fraction of the sample where $y = j$) and let $\hat{p}_j, j = 1, \ldots, m$, be the fitted frequencies. For example, in the Poisson model, $\hat{p}_j = \frac{1}{n}\sum_{i=1}^{n} \hat{\lambda}_i^j \exp(-\hat{\lambda}_i)/j!$, where $\hat{\lambda}_i = \sum_{k=1}^{p} \exp(x_i^k \hat{\beta}_k)$ is the expected count value for individual i.

To start with, one can compare $\bar{p}_j$ to $\hat{p}_j$ for specific values of the count variable j to gain some insight about the range of counts where the model has a tendency to over or under predict or to allow a visual inspection of the predictive performance of competing models. This computation can be done in **Countr** by a call to the function `compareToGLM()` which can take a fitted Poisson and (optionally) a negative binomial model and compare them to a number of fitted `renewal` models passed as additional arguments. The function returns a table with $\bar{p}_j$ (`Actual`) and the estimates $\hat{p}_j$ induced by the different models. The contribution to the Pearson statistic of each cell, defined as $\sum_{j=1}^{J} n\frac{(\bar{p}_j - \hat{p}_j)^2}{\bar{p}_j}$, is computed, as well. The result can be visualised by a call to `frequency_plot()`, see Section 4.

Formal tests are often used for model validation. Cameron and Trivedi (2013, Section 5.3.4) suggest a formal *chi-square goodness-of-fit test* which is a generalisation of the Pearson's *chi-square test* and controls for estimation error in $\hat{p}_j$. The test is a conditional moment test. Its gradient version, implemented in **Countr**, is justified for renewal models as they are fully parametric and parameter estimation is based on maximum-likelihood. The test is carried out by function `chiSq_gof()`.

Applications of the above tests are given in the following sections.

3. Package design

The **Countr** package is available from CRAN https://cran.r-project.org/package=Countr and can be installed using the standard R tools.

The main function in **Countr** is `renewalCount()`. It fits renewal regression models for count data using maximum likelihood. Several built-in count distributions are provided. The distributions are parameterised in terms of the corresponding inter-arrival times, see Table 1. The Poisson distribution is given in the table for reference and can be fitted using base R's `glm()`). User-defined distributions are also supported.

The `renewalCount()` function returns the fitted model as an object from S3 class `"renewal"`. The standard interface to the modelling functions is maintained, as much as possible. In particular, methods for `summary()`, `predict()`, `confint()`, `coef()` and similar functions are available, see also Table 7.

The **Countr** package also exports functions for the computation of the probabilities associated with several renewal count models. The probability computations are rather intensive and are mostly implemented in C++ with the help of the **RcppArmadillo** (Eddelbuettel and Sanderson 2014) package. Several methods are provided offering various degrees of trade-off between speed and accuracy, see Section 2.2.

Renewal regression models are fitted with the function `renewalCount()`. It has been designed to mimic the GLM functionality in R. In fact, users familiar with `glm()` should recognize several common arguments in `renewalCount()`'s interface:

```
R> renewalCount(formula, data, subset, na.action, weights, offset,
```

```
+     dist = c("weibull", "weibullgam", "custom", "gamma", "gengamma", "burr"),
+     anc = NULL, convPars = NULL, link = NULL, time = 1.0,
+     control = renewal.control(...), customPars = NULL,
+     seriesPars = NULL, weiMethod = NULL,
+     computeHessian = TRUE, model = TRUE, y = TRUE, x = FALSE, ...)
```

The first line contains the standard model-frame specifications, while arguments `computeHessian`, `model`, `x` and `y` are boolean flags indicating whether the returned object should contain the variance-covariance matrix, the model frame, the model matrix and the response, respectively. All default to `TRUE`. If `computeHessian` is `FALSE`, the variance-covariance matrix is not computed. The remaining arguments are specific to the renewal regression model.

The minimum required inputs are `formula` (an R formula), `data` (a data frame) and `dist` (a character string). Argument `formula` describes the model, `data` contains the values of the response and the covariates, while `dist` specifies the desired count model distribution.

The fitting process is based on maximum likelihood using optimization routines implemented in package **optimx** (Nash and Varadhan 2011). Users can customize different aspects of the fitting process and control what is returned but if the minimum inputs are provided the routine will work just fine. We give more details in the following sections. Additional guidance can be found in the package documentation and the vignettes.

4. Quick start - an example without covariates

The examples in this and later sections assume that the package is made available in the current session via

```
R> library("Countr")
```

We also need `dplyr` (Wickham and Francois 2016) and `xtable` (Dahl 2016), which provide usefull facilities for data manipulation and presentation:

```
R> library("dplyr")
R> library("xtable")
```

The purpose of this section is to give users a general sense of the package, including the components, what they do and some basic usage. For this purpose, we use the *football* dataset shipped with `Countr` which contains the final scores of the 1104 matches played in the English Premier League from season 2009/2010 to season 2016/2017 (380 matches per season). The game data and home and away team names are also provided. The data were collected from `http://www.football-data.co.uk/englandm.php` and slightly formatted and simplified.

As discussed in length in Kharrat (2016, Chapter 4) and more briefly in Boshnakov, Kharrat, and McHale (2017), the main issue with the Poisson model when modelling the goals scored by a team in football is that the hazard function (the instant probability of scoring) remains constant for every time unit (minutes say in football). However, empirical studies showed that this is rather questionable. In particular, goals are more likely to be scored at the end of each half because of players' tiredness, see for example Dixon and Robinson (1998, Figure 1).

Renewal-count distributions give the flexibility to drop the constant intensity assumption by selecting non-exponential interval-arrival distributions. One strategy to select this distribution

is discussed in Kharrat (2016, Chapter 4). Here we simply say that the Weibull density seemed to provide the best fit and will be used in this example.

We focus on the goals scored by the away team:

```
R> data("football")
R> table(football$awayTeamGoals)

   0    1    2    3    4    5    6    7
1028 1012  604  279   82   24   10    1
```

Our aim here is not to conduct an extensive analysis of the data but to highlight the improvement introduced by the Weibull-count model compared to Poisson.

We fit the Poisson model using `glm()` with the family argument set to `poisson`. For the Weibull-count model we use `renewalCount()` with `dist = "weibull"`. Both models are intercept only (no covariates specified).

```
R> away_poiss <- glm(formula = awayTeamGoals ~ 1, family = poisson,
+    data = football)
R> away_wei <- renewalCount(formula = awayTeamGoals ~ 1, data = football,
+    dist = "weibull", computeHessian = FALSE,
+    control = renewal.control(trace = 0))
```

We start by investigating the distribution of goals and the associated fitted probabilities induced by the two models. The away team rarely scores more than 4 goals and hence we decided to aggregate counts of 5 and larger.

```
R> breaks_ <- 0:5
R> pears <- compareToGLM(poisson_model = away_poiss, breaks = breaks_,
+    weibull = away_wei)
```

Figure 1 shows the observed relative frequencies together with the predictions from the two models.

As expected, the most likely outcome for away goals is 0, 1 and to some extent 2. Eyeballing Figure 1, the Weibull-count model is a clear improvement over the Poisson model.

We can now verify these findings with formal statistical tests. As discussed before, the two models are nested and the likelihood ratio test can be used to discriminate between them:

```
R> lr <- lmtest::lrtest(away_poiss, away_wei)
R> lr

Likelihood ratio test

Model 1: awayTeamGoals ~ 1
Model 2: awayTeamGoals ~ 1
  #Df LogLik Df Chisq Pr(>Chisq)
1   1  -4364
2   2  -4350  1  28.5    9.5e-08
```

```
R> library("dplyr")
R> frequency_plot(pears$Counts, pears$Actual,
+    dplyr::select(pears, contains("_predicted")),
+    colours = c("grey", "blue", "green", "black"))
```

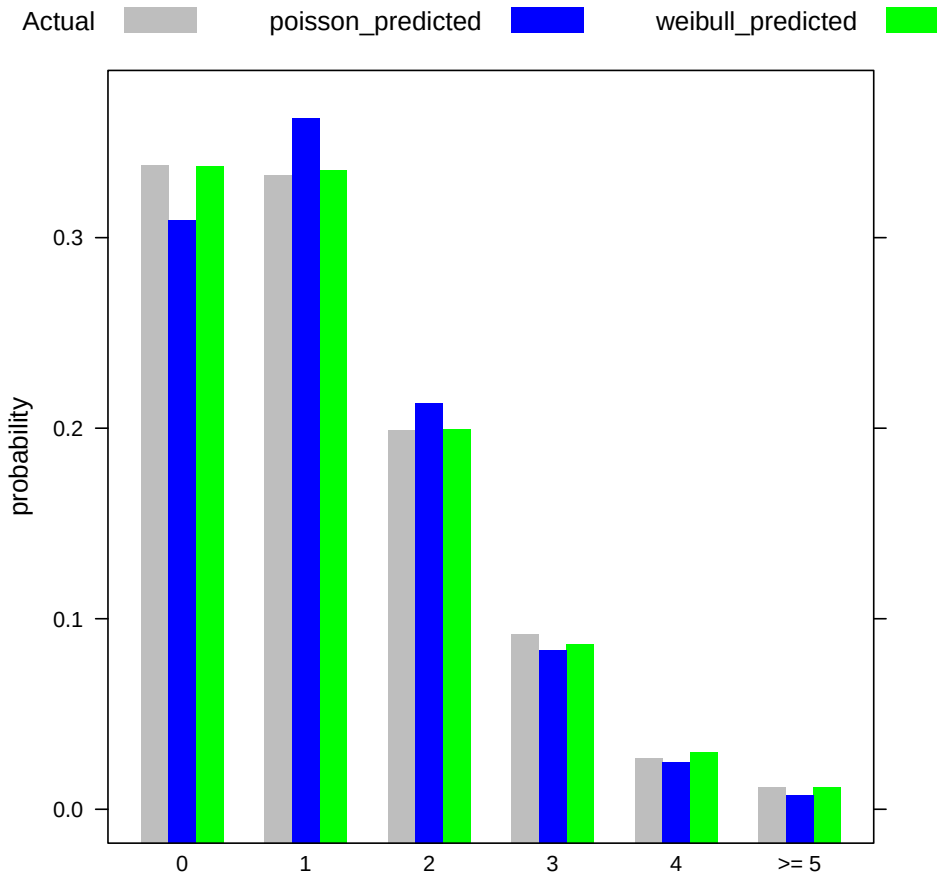

Figure 1: Comparison of Weibull and Poisson models for football data. The predicted frequencies from the Weibull model (green) match the observed frequencies (grey) better than those from the Poisson model (blue).

The likelihood ratio test's null hypothesis (both models present equal fits) is rejected at any reasonable confidence level, thus confirming that the Weibull addition induced by the shape parameter does improve the fitting.

Finally, formal chi-square goodness-of-fit tests implemented in `Countr::chiSq_gof()` can be used to judge how well the models describe the data. Here are the results for the Weibull-count and Poisson models:

```
R> gof_wei <- chiSq_gof(away_wei, breaks = breaks_)
R> gof_pois <- chiSq_gof(away_poiss, breaks = breaks_)
R> rbind(Poisson = gof_pois, "Weibull-count" = gof_wei)

chi-square goodness-of-fit test

Cells considered 0 1 2 3 4 >= 5
              DF Chisq Pr(>Chisq)
Poisson        5 27.60    4.4e-05
Weibull-count  5  7.26        0.2
```

The results from the two tests are printed together for convenience. The null hypothesis that the Weibull model is adequate cannot be rejected (p-value 0.2), supporting the claim that the Weibull-count model describes the data well. On the other hand, the hypothesis that the Poisson model is adequate is confidently rejected (p-value 4.35e-05).

5. An extended example with under-dispersion

We illustrate the usage of **Countr** with the fertility data, first described in Winkelmann (1995, Section 5) and re-analyzed by McShane *et al.* (2008) and Baker and Kharrat (2017). The `fertility` dataset contains information about a sample of 1,243 women who were over 44 years old in 1985 and answered the questions of the German Socio-Economic Panel. The responses are arranged in a data frame with one row for each mother and 9 columns[1], coded as follows:

- `children` — number of children.
- `german` — German nationality, a factor variable with levels `yes` and `no`.
- `years_school` — general education, measured as years of schooling.
- `voc_train`, `university` — post-secondary education: vocational training (`voc_train`) and university (`university`), factor variables with levels `yes` and `no`.
- `religion` — factor variable with levels `Catholic`, `Muslim`, `Protestant`, and `Other`.
- `rural` — rural, a factor variable with levels `yes` and `no`.
- `year_birth` — year of birth,

[1]The dataset is equivalent to the earlier references but, for convenience, we have renamed the variables and replaced the dummy variables with factors.

- `age_marriage` — age at marriage.

The dataset is available when **Countr** is attached. It can also be loaded independently using `data()`, e.g.,

```
R> data("fertility", package = "Countr")
```

The motivation to use the fertility data is twofold. First, we wanted to analyse a dataset with under-dispersion, where by construction the Poisson model and the natural extensions, such as negative binomial, fail to capture this aspect in the data. Second, we wanted to give the users the software to reproduce the results discussed in the reference papers Winkelmann (1995) and McShane *et al.* (2008).

The first few rows of the data are shown in Table 3. The response variable considered

	children	german	years_school	voc_train	university	religion	year_birth	rural	age_marriage
1	2	no	8	no	no	Catholic	42	yes	20
2	3	no	8	no	no	Catholic	55	yes	21
3	2	no	8	no	no	Catholic	51	yes	24
4	4	no	8	no	no	Catholic	54	no	26
5	2	no	8	no	no	Catholic	46	yes	22
6	2	no	8	no	no	Catholic	41	no	18

Table 3: First few rows of fertility data.

here is the number of children per woman (`children`). The average number of children observed in this sample is 2.384 and variance is 2.33, so there is no apparent under- or over-dispersion. However, when the additional variables are taken into account under-dispersion becomes evident as discussed by McShane *et al.* (2008) and confirmed by our analysis below. A frequency table of this variable is shown in Table 4. There are 8 possible explanatory

	0	1	2	3	4	5	6	7	8	>= 9
Frequency	76	239	483	228	118	44	30	10	8	7
Relative frequency	0.061	0.19	0.39	0.18	0.095	0.035	0.024	0.008	0.0064	0.0056

Table 4: Fertility data: Frequency distribution of column `children`.

variables: 3 numeric and 5 categorical (factors). Tables 5 and 6 show summaries of these variables.

5.1. Model specification

Specifying the count distribution

The count distribution is selected by specifying the distribution of the inter-arrival times. **Countr** currently provides the four built-in distributions discussed in Table 1. Besides, another

	german	voc_train	university	religion	rural
X	no :245	no :704	no :1207	Catholic :130	no :613
X.1	yes:998	yes:539	yes: 36	Muslim :502	yes:630
X.2				Other : 75	
X.3				Protestant:536	

Table 5: Summary of the factor variables

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
children	0.00	1.00	2.00	2.38	3.00	11.00
years_school	8.00	9.00	9.00	9.10	9.00	13.00
year_birth	40.00	45.00	50.00	51.99	58.00	83.00
age_marriage	17.00	21.00	23.00	23.11	25.00	30.00

Table 6: Summary of the numeric explanatory variables

distribution (`dist = "weibullgam"`) has been implemented. It is known as the Weibull-gamma (or compound Weibull) and is obtained from Weibull by letting the parameter λ have a gamma distribution. The Weibull-gamma count distribution has Weibull-gamma for its interarrival times. This model has been derived in details in McShane *et al.* (2008). This model can be seen as a means to model heterogeneity of individuals' inter-arrival times, the same way the negative binomial extends the Poisson model. We found this model to be numerically unstable and it should be used with care (see also the discussion in Baker and Kharrat 2017, Section 7.4).

For the `renewalCount()` function and other functions in the package that provide a choice, the desired inter-arrival distribution is specified by the argument `dist` as a character string, which should have one of the values reported in Table 2. Inter-arrival distributions defined by the user are also supported and specified by `dist = "custom"`, see Section 5.3.

Specifying covariates

Covariates can be introduced using familiar R formula syntax. In the examples with the `fertility` dataset we will use the following formula:

```
R> regModel <- children ~ german + years_school + voc_train + university +
+    religion + rural + year_birth + age_marriage
```

When supplied as argument `formula` in a call to `renewalCount()` the left-hand side of the formula specifies the response variable. The right-hand side gives the covariates for the linear relationship to the (possibly transformed by a link function) corresponding parameter of the count distribution.

Different links can be specified using the `link` argument of `renewalCount()`. It should be a named list, where the link for each parameter of the distribution is given as a character string. For example, the `log` link can be specified for the shape and scale parameters of the Weibull distribution as follows:

```
R> link_weibull <- list(scale = "log", shape = "log")
```

Possible options for the link function are `"log"`, `"cauchit"`, `"cloglog"`, `"probit"`, `"logit"` and `"identity"` (default for user defined distributions).

5.2. Fitting built-in models

The fitting function `renewalCount()` has many arguments but when fitting models with the built-in count distributions it is usually sufficient to specify the model, the data, and possibly initial values, leaving the remaining settings to their default values. For example, the gamma model of Winkelmann (1995) can be fitted as follows:

```
R> gamModel <- renewalCount(formula = regModel, data = fertility,
+    dist = "gamma", control = renewal.control(trace = 0) )
```

The setting `trace = 0` in this and other examples prevents the optimising function from printing during the optimisation step.

Almost any aspect of the computation can be customized in `renewalCount()` and options are provided to give the user control over the computation of the initial values, the numerical optimization algorithm, the method for computing the count probability and the returned values, among others.

User defined initial values

As usual in non-linear optimisation, for best results informed initial values should be provided whenever possible.

One strategy is to fit a Poisson GLM model and use its parameter estimates as starting values for the linear predictor of the location parameter. Since the models fitted by **Countr** contain additional parameters, these need to be set suitably. For example, the Weibull count model reduces to the Poisson if the shape parameter is equal to one, so this is a natural initial value for this parameter. More generally, for a count distribution which generalizes the Poisson model, values of the parameters that reduce the distribution to Poisson are often suitable starting points. Some theoretical results support this procedure, see for example Cameron and Trivedi (2013, Section 3.2).

The above strategy is adopted by `renewalCount()` when no initial values are provided. However, when fitting a model with more than one linear predictor, initial values are required from the user.

The initial values are passed to `renewalCount()` as a named numeric vector. For this, the names of the coefficients are needed. They have the form `par_covname`, where `covname` is the name of a covariate and `par` is the name of the distribution parameter, such as `shape` to which the covariate is linked. Intercepts are named `par_`. The names of the distribution parameters can be found by a call to `getParNames()`. For example, this shows that the parameters of the Weibul distribution are named `"scale"` and `"shape"`:

```
R> getParNames("weibull")

[1] "scale" "shape"
```

A suitably named vector can be obtained by extracting the coefficients of an existing model and, if necessary, changing their values. The convenience function `renewalCoef(object,`

`target)` eliminates most of the tedious work by going even further — it takes `object`, usually a fitted model, extracts coefficients, and renames them for use with the parameter or distribution specified by target, see the examples below. If a suitable model is not available, the function `renewalNames()` can be used to get a character vector of names of parameters. It can be used in two ways. The first, `renewalNames(object, target)`, has similar semantics to `renewalCoef()`. The second is `renewalNames(object,...)`, where `"..."` are the same arguments that would be used in a call to `renewalCount()`. In this case `renewalNames()` returns the names of the coefficients of the model that would be produced if `renewalCount()` was called with the same parameters. For example, changing `renewalCount` to `renewalNames` in the code used to obtain `gamModel`, we get:

```
R> renewalNames(regModel, data = fertility, dist = "gamma")
```

```
 [1] "rate_"                  "rate_germanyes"
 [3] "rate_years_school"      "rate_voc_trainyes"
 [5] "rate_universityyes"     "rate_religionMuslim"
 [7] "rate_religionOther"     "rate_religionProtestant"
 [9] "rate_ruralyes"          "rate_year_birth"
[11] "rate_age_marriage"      "shape_"
```

We illustrate below the preparation of initial values for the Weibull count model of McShane *et al.* (2008). This is a model with one linear predictor, as in the Poisson model. An example for the case with more linear predictors is given later along with the discussion of regression on ancilliary parameters.

As discussed above, we fit a Poisson model:

```
R> IV <- glm(regModel, family = poisson(), data = fertility)
```

```
R> coef(IV)
```

```
      (Intercept)          germanyes       years_school       voc_trainyes
          1.14744            -0.20036             0.03351            -0.15278
   universityyes     religionMuslim      religionOther religionProtestant
         -0.15483             0.21804             0.54757             0.11341
        ruralyes         year_birth       age_marriage
          0.05907             0.00242            -0.03045
```

We then rename the coefficients of model `IV` to link them to the first parameter, `scale`, of the Weibull distribution:

```
R> startW <- renewalCoef(IV, target = "scale")
```

The Poisson model is a particular case of the Weibull model with shape parameter equal to one, which we use as a natural initial value for this parameter and append it to `startW` to complete it. Note that the regression is done on `log()` scale for both the shape and scale parameters as explained above.

```
R> startW <- c(startW, "shape_" = log(1))
R> startW
```

```
                scale_          scale_germanyes       scale_years_school
              1.14744                 -0.20036                  0.03351
     scale_voc_trainyes      scale_universityyes     scale_religionMuslim
             -0.15278                 -0.15483                  0.21804
    scale_religionOther scale_religionProtestant          scale_ruralyes
              0.54757                  0.11341                  0.05907
       scale_year_birth       scale_age_marriage                   shape_
              0.00242                 -0.03045                  0.00000
```

Finally, we fit the model, The initial values are passed to `renewalCount()` through the `renewal.control()` routine that will run a sanity check before passing them to the optimizer:

```
R> weiModel <- renewalCount(formula = regModel, data = fertility,
+    dist = "weibull", control = renewal.control(trace = 0, start = startW))
```

This model is further discussed in Section 5.4.

Customizing the optimization routine

As mentioned above, `renewalCount()` maximizes the log-likelihood of the desired model by a call to `optimx()` from package **optimx** (Nash and Varadhan 2011). The default is to use `method = "nlminb"` with a maximum of 1000 iterations. Users can change this again through the `renewal.control()` routine. Any other option accepted by `optimx()` can also be passed in `renewal.control()`, e.g.,

```
R> weiModelA <- renewalCount(formula = regModel, data = fertility,
+    dist = "weibull",
+    control = renewal.control(trace = 0, method = "L-BFGS-B"))
```

It is also possible to experiment with more than one optimisation algorithm in a single call to `renewalCount()`. The result with the highest value of the objective function (largest log-likelihood) will be preferred. This is illustrated below:

```
R> weiModel_many <- renewalCount(formula = regModel, data = fertility,
+    dist = "weibull", control = renewal.control(trace = 0,
+    method = c("nlminb", "Nelder-Mead", "BFGS")))
```

Field `optim` of the result contains a data frame giving, for each method, the estimates of the parameters, the run time (`xtimes`), the value of the log-likelihood evaluated at the estimates (`value`) and other quantities from the optimisation routine. In our example:

```
R> t(weiModel_many$optim)
```

```
                           nlminb      BFGS Nelder-Mead
scale_                   1.40e+00  1.41e+00    1.22e+00
scale_germanyes         -2.23e-01 -2.20e-01   -2.13e-01
scale_years_school       3.85e-02  3.90e-02    4.72e-02
scale_voc_trainyes      -1.73e-01 -1.75e-01   -1.83e-01
scale_universityyes     -1.81e-01 -1.85e-01   -2.25e-01
scale_religionMuslim     2.42e-01  2.42e-01    2.26e-01
scale_religionOther      6.39e-01  6.39e-01    6.38e-01
scale_religionProtestant 1.23e-01  1.24e-01    9.78e-02
scale_ruralyes           6.81e-02  6.77e-02    6.15e-02
scale_year_birth         2.30e-03  1.95e-03    2.13e-03
scale_age_marriage      -3.40e-02 -3.40e-02   -2.86e-02
shape_                   2.12e-01  2.11e-01    2.09e-01
value                   -2.08e+03 -2.08e+03   -2.08e+03
fevals                   1.05e+02  9.50e+01    1.00e+03
gevals                   7.92e+02  1.50e+01          NA
niter                    5.60e+01        NA          NA
convcode                 0.00e+00  0.00e+00    1.00e+00
kkt1                           NA        NA          NA
kkt2                           NA        NA          NA
xtimes                   2.31e+01  1.15e+01    2.58e+01
```

The algorithms are arranged in decreasing order of the likelihood. Here the three algorithms found a similar value of the log-likelihood but the `"Nelder-Mead"` did not converge. Although substantially slower, the `"nlminb"` was preferred because it gave a sligltly higher likelihood (-2077.022, for `"nlminb"` compared to -2077.034, for the `"BFGS"` algorithm).

Regression models on the ancillary parameters

So far we have given examples of regression models in which the parameter regressed on is the location parameter, or more precisely, the first parameter of the count distribution. **Countr** offers the possiblity to specify covariates on the "ancillary" parameters (the ones that determine the shape, the variance or other higher moments). This can be done using either argument `anc` of `renewalCount()` or an extended formula syntax.

If `anc` is supplied, it should be a named list in which each component is a model formula describing a regression equation for an ancillary parameter. Only the right-hand sides of these formulas are used. This setup is modelled on (and is compatible with) package `flexsurv` (Jackson 2016, function `flexsurvreg()`). For example, to specify a Weibull count model with response variable `y` and covariates `x1`, `x2` and `x3`, we could use something like `formula = y ~ x1 + x2 + x3` and `anc = list(shape = ~x1)` in the call to `renewalCount()` to request different covariates for the scale and shape parameters.

Alternatively, the regression terms for all parameters can be put on the right-hand side (rhs) of argument `formula`, separated by `|`. In this case the left-hand side (lhs) may also contain terms separated by `|` to designate the response variable and the distribution parameters. The latter can be omitted if there is no ambiguity. For example, the same model as above can be specified with `formula = y ~ x1 + x2 + x3 | x1`, or more verbosely by `formula = y | shape ~ x1 + x2 + x3 | x1`. In the latter case the i-th term on the lhs is paired with the

i-th term on the rhs. If the same model formula is desired for all parameters, there is no need to repeat the rhs. Thus, `formula = y | shape ~ x1` requests `~ x1` for both, the response variable and the shape parameter. Note that `x1` applies only to terms listed in the lhs—if `shape` is omitted, `formula = y ~ x1`, `x1` will not be used for it.

Formulas can be created and manipulated with standard R tools and `Formula::as.Formula`. Sometimes it may be troublesome to manipulate long formulas, so package **Countr** provides the helper function `CountrFormula` as an aid in creating formulas suitable for `renewalCount`. Here is an illustrative example of its use:

```
R>   CountrFormula(y ~ x1 + x2 + x3, shape = ~x1)

y | shape ~ x1 + x2 + x3 | x1
```

We show below an example of fitting a model for the generalized gamma inter-arrival times with covariates applied to all the distribution parameters. The names of the distribution parameters in this case are `"mu"` (location), `"sigma"` and `"Q"` (see Table 2). To fit a model using the same model formula, `regModel`, for all parameters we can call `renewalCount` with `formula = regModel` and `anc` set up as follows:

```
R> anc <- list(sigma = regModel, Q = regModel)
```

In the ancillary regression case, starting values have to be provided. As discussed earlier, we obtain informative initial values for the location parameter from a Poisson model. Here we can use the previously model `IV`. We also set the intercepts for (log) `"sigma"` and `"Q"` to one and the remaining regression coefficients to zero. With the help of `renewalCoef()` all this can be done with a couple of lines:

```
R> startA <- renewalCoef(IV, target = "gengamma")
R> startA[c("Q_", "sigma_")] <- c(1, log(1))
R> startA
```

```
                    mu_             mu_germanyes         mu_years_school
                1.14744                  -0.20036                  0.03351
         mu_voc_trainyes        mu_universityyes       mu_religionMuslim
                -0.15278                  -0.15483                  0.21804
        mu_religionOther    mu_religionProtestant             mu_ruralyes
                 0.54757                   0.11341                  0.05907
           mu_year_birth         mu_age_marriage                   sigma_
                 0.00242                  -0.03045                  0.00000
         sigma_germanyes       sigma_years_school       sigma_voc_trainyes
                 0.00000                   0.00000                  0.00000
     sigma_universityyes     sigma_religionMuslim      sigma_religionOther
                 0.00000                   0.00000                  0.00000
sigma_religionProtestant           sigma_ruralyes         sigma_year_birth
                 0.00000                   0.00000                  0.00000
      sigma_age_marriage                        Q_             Q_germanyes
                 0.00000                   1.00000                  0.00000
```

```
      Q_years_school          Q_voc_trainyes          Q_universityyes
             0.00000                 0.00000                  0.00000
    Q_religionMuslim        Q_religionOther    Q_religionProtestant
             0.00000                 0.00000                  0.00000
         Q_ruralyes           Q_year_birth          Q_age_marriage
             0.00000                 0.00000                  0.00000
```

The above illustrates the use of the name of a count distribution for argument `target`. In that case `renewalCoef()` assumes that the same formula is used for all parameters. The model now can be fit with:

```
R> gengamModel_ext0 <- renewalCount(formula = regModel, data = fertility,
+    dist = "gengamma", anc = anc,
+    control = renewal.control(start = startA, trace = 0),
+    computeHessian = FALSE)
```

To illustrate the use of different formulas for the different parameters, let us use `regModel` for the first parameter and the following specifications for the remaining parameters:

```
R> sigmaModel <- ~ german + university + religion + age_marriage
R> QModel     <- ~ german + religion + age_marriage
```

For use with argument `anc`, we create the following list:

```
R> anc <- list(sigma = sigmaModel, Q = QModel)
```

For the alternative formula method we prepare an extended formula:

```
R> regModelSQ <- Formula::as.Formula(regModel, sigmaModel, QModel)
```

We could use also

```
R> CountrFormula(regModel, sigma = sigmaModel, Q = QModel)

children | sigma | Q ~ german + years_school + voc_train + university +
    religion + rural + year_birth + age_marriage | german + university +
    religion + age_marriage | german + religion + age_marriage
```

As discussed earlier, we can obtain informative initial values for the location parameter from a Poisson model. Here we can use the previously fitted model `IV` for the location parameter. Similarly, for the other parameters we fit Poisson models with the respective formulas. Some justification for this comes from the properties of the Poisson distribution (most notably its variance is equal to the mean). Also, we use `update()` to set the response variable, since `sigmaModel` and `QModel` have empty left-hand sides:

```
R> IV2 <- glm(update(sigmaModel, children ~ .),
+    family = poisson(), data = fertility)
R> IV3 <- glm(update(QModel, children ~ .),
+    family = poisson(), data = fertility)
```

We construct the initial values from the above fits with the help of `renewalCoef()` and suitable settings for its argument `target`:

```
R> startGG <- c(renewalCoef(IV, target = "mu"),
+    renewalCoef(IV2, target = "sigma"),
+    renewalCoef(IV3, target = "Q"))
R> startGG
```

```
                  mu_             mu_germanyes         mu_years_school
             1.14744                  -0.20036                  0.03351
      mu_voc_trainyes        mu_universityyes       mu_religionMuslim
            -0.15278                  -0.15483                  0.21804
     mu_religionOther   mu_religionProtestant             mu_ruralyes
             0.54757                   0.11341                  0.05907
       mu_year_birth         mu_age_marriage                   sigma_
             0.00242                  -0.03045                  1.49635
     sigma_germanyes     sigma_universityyes   sigma_religionMuslim
            -0.20508                  -0.02061                  0.22214
sigma_religionOther sigma_religionProtestant      sigma_age_marriage
             0.56081                   0.11525                  -0.02848
                  Q_              Q_germanyes        Q_religionMuslim
             1.49871                  -0.20562                  0.22266
     Q_religionOther     Q_religionProtestant          Q_age_marriage
             0.56086                   0.11580                  -0.02860
```

This fits the model using argument `anc`:

```
R> fm_gengamAnc <- renewalCount(formula = regModel, data = fertility,
+    dist = "gengamma", anc = anc,
+    control = renewal.control(start = startGG, trace = 0),
+    computeHessian = FALSE)
```

Equivalently, the same results are obtained with the extended formula argument:

```
R> fm_gengam <- renewalCount(formula = regModelSQ, data = fertility,
+    dist = "gengamma",
+    control = renewal.control(start = startGG, trace = 0),
+    computeHessian = FALSE)
```

If the optimisation didn't converge or for other reasons, we may need to fit the model again. Here we want to fit the same model using the coefficients of the previous fit as initial values, so we can directly use `fm_gengam`. We also request the `spg` algorithm (Varadhan and Gilbert 2009) for this second iteration:

```
R> startBB <- coef(fm_gengam)
R> fm_gengam_ext <- renewalCount(formula = regModelSQ, data = fertility,
+    dist = "gengamma",
+    control = renewal.control(method = "spg", start = startBB, trace = 0),
+    computeHessian = FALSE )
```

A check of the convergence status flag reveals that the optimization did converge:

```
R> fm_gengam_ext$converged

[1] TRUE
```

5.3. Custom inter-arrival distributions

Instead of using the built-in distributions in `Countr`, users can also specify their own inter-arrival parametric distributions. For this to work, the following information is required:

- names of the distribution parameters, a character vector.
- survival distribution function, a function with signature `function(tt, distP)`, where `tt` is a vector of class `"numeric"` of non-negative values and `distP` gives the values of the distribution parameters as a named list.
- the name(s) of the link function(s); different link functions may be used for the different parameters. If not specified, `identity` will be used.

The Weibull inter-arrival distribution is one of the built-in distributions but as an illustrative example here is how it could be specified as a custom distribution:

```
R> parNames <- c("scale", "shape")
R> sWei <- function(tt, distP) exp( -distP[["scale"]] * tt ^ distP[["shape"]])
R> link <- list(scale = "log", shape = "log")
```

Here `parNames` defines the names of the parameters, `"sWei"` computes the survival distribution function and `link` requests `log` link for both parameters (a common choice for positive parameters).

Initial values are very desirable for user defined distributions and are computed as discussed in Section 5.2.1. We are going to fit the same model, so we can use the initial values `startW` defined in that section.

```
R> control_custom <- renewal.control(start = startW, trace = 0)
```

For custom inter-arrival distributions, convolution based methods are the only option. If the user is willing to speed up the computation using a Richardson correction scheme, the appropriate correction function that computes the correction coefficients must be passed. As argued by Baker and Kharrat (2017, Section 3.5), the appropriate values for the Weibull case are $(2, \alpha)$, where α is the shape parameter. This can be communicated to `renewalCount()` by defining a function whose argument is a named list of the distribution parameters, as in:

```
R> .getExtrapol <- function(distP) {
+    c(2, distP[["shape"]])
+  }
```

The names of the parameters, the survival function and the extrapolation parameters are passed to `renewalCount()` through argument `customPars`. In our example these are `parNames`, `sWei` and `.getExtrapol`, respectively. This illustrates the syntax for preparing the list:

```
R> customPars <- list(parNames = parNames, survivalFct = sWei,
+    extrapolFct = .getExtrapol)
```

There is also an argument for the links. A model with our custom specified distribution can now be fitted with:

```
R> weiModelCust <- renewalCount(formula = regModel, data = fertility,
+    dist = "custom", link = link, control = control_custom,
+    customPars = customPars, computeHessian = FALSE)
```

Note that the computations in this example can be much slower than for the equivalent built-in distribution (that is why the Hessian computation has been turned off), since the crucial parts of the latter are implemented in C++. Therefore, we recommend using built-in distributions as much as possible and simply consider custom inter-arrival distributions for exploratory purpose where long computation time is not an issue.

5.4. Working with the fitted models

The function `renewalCount()` produces an S3 object from class `"renewal"`. Methods for a number of R functions are provided, so that objects from class `"renewal"` can be manipulated and explored in a familiar way. Table 7 lists generic functions from base R with methods for objects from class `"renewal"`. Only functions with explicitly defined renewal methods are listed. The default methods for generics without such methods may also work, when they access properties of objects via calls to functions such as `coef` to collect the information they need.

`coef()`	`fitted()`	`df.residual()`	`print()`
`confint()`	`predict()`	`extractAIC()`	`summary()`
`vcov()`	`residuals()`	`df.residual()`	
	`model.matrix()`	`logLik()`	
	`nobs()`		

Table 7: Generic functions from base R with methods for objects from class `"renewal"`.

Model fit

The usual `summary()` method can be used to check the coefficients' estimates together with their standard errors (computed using numerical estimates of the gradient and Hessian) and Wald test statistics. Here is a summary of Winkelmann's model fitted above:

```
R> summary(gamModel)

Call:
renewalCount(formula = regModel, data = fertility, dist = "gamma",
```

```
    control = renewal.control(trace = 0))

Pearson residuals:
    Min      1Q  Median      3Q     Max
-2.6410 -0.7262 -0.0901  0.4890  6.7411
Inter-arrival dist.: gamma
              Links: rate: link log, shape: link log

Count model coefficients
                        Estimate Std. Error z value Pr(>|z|)
rate_                    1.55671    0.25234    6.17  6.9e-10
rate_germanyes          -0.18976    0.05904   -3.21  0.00131
rate_years_school        0.03168    0.02650    1.20  0.23187
rate_voc_trainyes       -0.14393    0.03584   -4.02  5.9e-05
rate_universityyes      -0.14605    0.12961   -1.13  0.25982
rate_religionMuslim      0.20577    0.05780    3.56  0.00037
rate_religionOther       0.52263    0.06984    7.48  7.3e-14
rate_religionProtestant  0.10714    0.06230    1.72  0.08546
rate_ruralyes            0.05549    0.03119    1.78  0.07519
rate_year_birth          0.00234    0.00195    1.20  0.22862
rate_age_marriage       -0.02880    0.00533   -5.41  6.5e-08
shape_                   0.36417    0.04937    7.38  1.6e-13

Number of iterations in nlminb optimization: 66

Execution time   94
Log-likelihood: -2078.226 on 12 Df
```

The results are exactly the same as the ones in Winkelmann (1995, Table 1)[2]. Similarly, the results for `weiModel` below coincide with those given by McShane *et al.* (2008, Table 2)[3]:

```
R> summary(weiModel)

Call:
renewalCount(formula = regModel, data = fertility, dist = "weibull",
    control = renewal.control(trace = 0, start = startW))

Pearson residuals:
    Min      1Q  Median      3Q     Max
-2.6616 -0.7300 -0.0932  0.4978  6.7361
Inter-arrival dist.: weibull
```

[2]Note that the regression model defined in Winkelmann (1995, Equation (17)) is slightly different and hence the constant value defined in Table 1 is related to our estimated `rate_` parameter by $exp(Constant) * \alpha = exp(scale\_)$.

[3]The value of λ in McShane *et al.* (2008, Table 2) is the exponential of the value of `scale_`. The same applies for the value of `shape_`. Also note that the standard errors in their table are obtained by the bootstrap procedure with 100 replicates.

```
              Links: scale: link log, shape: link log

Count model coefficients
                        Estimate Std. Error z value Pr(>|z|)
scale_                   1.39722    0.31425    4.45  8.7e-06
scale_germanyes         -0.22255    0.07181   -3.10  0.00194
scale_years_school       0.03853    0.03269    1.18  0.23854
scale_voc_trainyes      -0.17335    0.04395   -3.94  8.0e-05
scale_universityyes     -0.18146    0.16032   -1.13  0.25771
scale_religionMuslim     0.24200    0.07018    3.45  0.00056
scale_religionOther      0.63875    0.08670    7.37  1.7e-13
scale_religionProtestant 0.12314    0.07554    1.63  0.10305
scale_ruralyes           0.06806    0.03812    1.79  0.07420
scale_year_birth         0.00230    0.00230    1.00  0.31624
scale_age_marriage      -0.03403    0.00635   -5.36  8.5e-08
shape_                   0.21200    0.02720    7.79  6.5e-15

Number of iterations in nlminb optimization: 56

Execution time   23
Log-likelihood: -2077.022 on 12 Df
```

Not surprisingly, the summary shows that religion has a major impact on the number of children, as are being a German and vocational training. On the other hand, university education seems not significant (p-value 0.2577), even though the effect (the value, -0.1815, of the coefficient) is almost the same as for vocational training. Similarly, years of schooling seems not significant. These variables are highly confounded, so such conclusions should not be taken at face value. Note also that the shape parameter is more than seven times greater than its standard error, so we have strong evidence that $\log(\beta) > 0$, i.e., $\beta > 1$, which corresponds to under-dispersion.

Bootstrap standard errors

Standard errors and other quantities computed from the Hessian are based on assumptions, which are often difficult to check. Non-parametric bootstrap is often a useful alternative. Another alternative would be to consider sandwich-type standard errors. The current version of **Countr** (3.5.0) only implements the boostrap option and the sandwich alternative will be added in the next update. The bootstrap computations are based on function `boot()` from the package with the same name (Canty and Ripley 2017). Bootstrap confidence intervals rely on additional methods implemented in `confint.boot()` from package `car` (Fox and Weisberg 2011).

Bootstrap standard errors and confidence intervals can be computed by setting `type = "boot"` and specifying the desired number of bootstrap samples with argument `R`. Cameron and Trivedi (2013, Section 2.6.4) observe that 400 bootstrap samples are often enough. This can be slow, so here we give an example with $R = 5$ replicates for illustration:

```
R> se_boot <- se.coef(object = weiModel, type =  "boot", R = 5)
R> confint_boot <- confint(object = weiModel, type = "boot", R = 5)
```

For `confint()`, the type of bootstrap confidence interval can be specified with argument `bootType`, which can be one of `"norm"` (normal approximation, the default), `basic` (basic bootstrap), `percent` (percentile method) and `"bca"` (bias-corrected and accelerated method), see Davison and Hinkley (1997, Chapter 5) for details on the methods.

In the above example, the computation of `se_boot` and `confint_boot` involves separately generated bootstrap samples. This can be slow with realistic number of bootstrap samples. Also, it may be desirable to do such related calculations on the same bootstrap sample. This can be achieved by creating a bootstrap sample separately, using `addBootSampleObject()`, which stores the bootstrap sample in component `"boot"` of the supplied model object. Renewal methods for functions like `confint` use the bootstrap sample if they find it there. For example, we could[4] do the following:

```
weiWithBoot <- addBootSampleObject(weiModel, R = 400, parallel = "multicore",
  ncpus = 14)
```

We use arguments `parallel` and `ncpus` to speed up the computations by using parallel processing facilities provided by `boot::boot`, see `help(boot::boot)`. Now these computations would use the stored bootstrap sample in `weiWithBoot`:

```
se_boot <- se.coef(object = weiWithBoot, type = "boot")
confint_boot <- confint(object = weiWithBoot, type = "boot")
```

To use another bootstrap sample, set component `"boot"` of `weiWithBoot` to `NULL` or remove it. Vignette `vignette("VarianceCovariance", package="Countr")` gives further examples and details.

Prediction

Predictions from the fitted model are obtained by calling `predict()`. Two types of prediction are available: predicting a given count (response) value (`type = "prob"`), i.e., the probability of observing a specific value of the count variable (the value of `Y` in our `data.frame`) given the (individual) covariates or predicting the expected value (`type = "response"`).

The procedure is illustrated for the first few individuals in the `fertility` data:

```
R> newData <- head(fertility)

R> predNew.response <- predict(weiModel, newdata = newData, type = "response",
+    se.fit = TRUE)
R> predNew.prob <- predict(weiModel, newdata = newData, type = "prob",
+    se.fit = TRUE)

R> predtable <- data.frame(newData$children, predNew.prob$values,
+    predNew.response$values)
R> names(predtable) <- c("Y", "P(Y=y|x)", "E(Y|x)")
R> predtable
```

[4]This and the following two R commands are not presented as R code since the parallel processing options are system dependent.

```
  Y P(Y=y|x) E(Y|x)
1 2 0.284675 2.6356
2 3 0.253154 2.6250
3 2 0.303078 2.3843
4 4 0.095873 2.1319
5 2 0.295154 2.5040
6 2 0.285128 2.6303
```

The covariates are not printed here since they were shown previously in Table 3.

To conclude this section, we verify that the results produced by the built-in model and the user defined Weibull model are identical (up to rounding errors):

```
R> cbind(builtIn = coef(weiModel), user = coef(weiModelCust))
```

```
                         builtIn    user
scale_                    1.3972  1.3973
scale_germanyes          -0.2225 -0.2226
scale_years_school        0.0385  0.0385
scale_voc_trainyes       -0.1734 -0.1734
scale_universityyes      -0.1815 -0.1815
scale_religionMuslim      0.2420  0.2420
scale_religionOther       0.6388  0.6388
scale_religionProtestant  0.1231  0.1231
scale_ruralyes            0.0681  0.0681
scale_year_birth          0.0023  0.0023
scale_age_marriage       -0.0340 -0.0340
shape_                    0.2120  0.2121
```

6. Model selection and comparison

In the previous section several models were fitted to the fertility data. It is natural to ask the question: Which model presents the best fit to the data and hence which one should be preferred? A strategy has been described in Section 2.1 and is inspired from the demand for medical care example detailed in Cameron and Trivedi (2013, Section 6.3). It is illustrated here with a real data example.

The dataset used in this example is the `quine` data from package `MASS` (Venables and Ripley 2002), first analysed by Aitkin (1978).

```
R> data("quine", package = "MASS")
```

The dataset gives the number of days absent from school (`Days`) of 146 children in a particular school year. A number of explanatory variables are available describing the children's ethnic background (`Eth`), sex (`Sex`), age (`Age`) and learner status (`Lrn`). The count variable `Days` is characterised by large *overdispersion* — the variance is more than 16 times larger the mean, 264.2 versus 16.46. Table 8 gives an idea about its distribution. The entries in the table were calculated as follows:

```
R> breaks_ <- c(0, 1, 3, 5:7, 9, 12, 15, 17, 23, 27, 32)
R> freqtable <-
+    count_table(count = quine$Days, breaks = breaks_, formatChar = TRUE)
```

	0	1-2	3-4	5	6	7-8	9-11
Frequency	9	10	7	19	8	10	13
Relative frequency	0.062	0.068	0.048	0.13	0.055	0.068	0.089

	12-14	15-16	17-22	23-26	27-31	>= 32
Frequency	13	6	14	6	6	25
Relative frequency	0.089	0.041	0.096	0.041	0.041	0.17

Table 8: quine data: Frequency distribution of column `Days`.

Given the extreme over-dispersion observed in the `quine` data, we do not expect the Poisson model to perform well. Nevertheless, we can still use it as a starting point and treat it as a benchmark (any model worse than Poisson should be strongly rejected). We also consider the negative binomial (as implemented in `MASS::glm.nb()`) and 3 renewal-count models based on Weibull, gamma and generalised-gamma inter-arrival times. This gives a total of five models to choose from. The following code fits the 5 models:

```
R> quine_form <- as.formula(Days ~ Eth + Sex + Age + Lrn)
R> pois <- glm(quine_form, family = poisson(), data = quine)
R> nb <- MASS::glm.nb(quine_form, data = quine)
R> ## various renewal models
R> wei <- renewalCount(formula = quine_form, data = quine, dist = "weibull",
+    computeHessian = FALSE, weiMethod = "conv_dePril",
+    control = renewal.control(trace = 0))
R> gam <- renewalCount(formula = quine_form, data = quine, dist = "gamma",
+    computeHessian = FALSE, control = renewal.control(trace = 0))
R> gengam <- renewalCount(formula = quine_form, data = quine, dist = "gengamma",
+    computeHessian = FALSE, control = renewal.control(trace = 0))
```

The models considered here are fully parametric. Therefore, a straightforward method to discriminate between them is a likelihood ratio (LR) test. This is possible when models are nested and in this case the LR statistic has the usual $\chi^2(p)$ distribution, where p is the difference in the number of parameters in the models. Here we compare all renewal-count models against Poisson, negative-binomial against Poisson, Weibull-count against generalised-gamma and gamma against the generalised-gamma.

For non-nested models, the standard approach is to use information criteria such as the Akaike information criterion (AIC) and the Bayesian information criterion (BIC). This method can be applied to discriminate between Weibull and gamma renewal count models, and between these two models and the negative binomial.

Therefore, a possible strategy (similar to what has been suggested in Cameron and Trivedi (2013, Section 6.3.4 p233)) can be summarised as follows:

- Use the LR test to compare Poisson with negative binomial.

- Use the LR test to compare Poisson with Weibull-count.
- Use the LR test to compare Poisson with gamma-count.
- Use the LR test to compare Poisson with generalised-gamma-count.
- Use the LR test to compare Weibull-count with generalised-gamma-count.
- Use the LR test to compare gamma-count with generalised-gamma-count.
- Use information criteria to compare gamma-count with Weibull-count.
- Use information criteria to compare Weibull-count to negative binomial.

Here is the code for these tests [5]:

```
R> library("lmtest")
R> pois_nb <- lrtest(pois, nb)
R> pois_wei <- suppressWarnings(lrtest(pois, wei))
R> pois_gam <- suppressWarnings(lrtest(pois, gam))
R> pois_gengam <- suppressWarnings(lrtest(pois, gengam))
R> pois_res <- data.frame("Alternative model" =
+    c("negative-binomial", "weibull", "gamma", "generalised-gamma"),
+    Chisq = c(pois_nb$Chisq[2], pois_wei$Chisq[2],
+              pois_gam$Chisq[2], pois_gengam$Chisq[2]),
+    Df = c(1, 1, 1, 2),
+    Critical_value = c(rep(qchisq(0.95, 1), 3), qchisq(0.95, 2)),
+    stringsAsFactors = FALSE)
```

The results are summarised in Table 9. As observed in Table 9, the LR test rejects the null hypothesis and all the alternative models are preferred to Poisson. This is due to the large over-dispersion.

	Alternative.model	Chisq	Df	Critical_value
1	negative-binomial	1192.03	1.00	3.84
2	weibull	1193.21	1.00	3.84
3	gamma	1193.36	1.00	3.84
4	generalised-gamma	1194.46	2.00	5.99

Table 9: LR results against Poisson model. Each row compares an alternative model vs the Poisson model. All alternatives are preferable to Poisson. The critical value corresponds to a significance level of 5%

Next, we carry out LR test to discriminate between the renewal count models (see Table 10 for the results):

[5] The `suppressWarnings()` command is used to avoid printing a message to complain about the model objects being from different class.

```
R> gengam_wei <- lrtest(wei, gengam)
R> gengam_gam <- lrtest(gam, gengam)
R> gengam_res <- data.frame(Model = c("weibull", "gamma"),
+    Chisq = c(gengam_wei$Chisq[2], gengam_gam$Chisq[2]), Df = 1,
+    Critical_value = rep(qchisq(0.95, 1), 2), stringsAsFactors = FALSE)
```

	Model	Chisq	Df	Critical_value
1	weibull	1.25	1.00	3.84
2	gamma	1.10	1.00	3.84

Table 10: LR results against generalised-gamma model

The results in Table 10 suggest that the Weibull and gamma models should be preferred to the generalised gamma model.

Finally, we use information criteria to choose the best model among the Weibull and gamma renewal models and the negative binomial:

```
R> ic <- data.frame(Model = c("gamma", "weibull", "negative-binomial",
+    "generalised-gamma", "Poisson"),
+    AIC = c(AIC(gam), AIC(wei), AIC(nb), AIC(gengam), AIC(pois)),
+    BIC = c(BIC(gam), BIC(wei), BIC(nb), BIC(gengam), BIC(pois)),
+    stringsAsFactors = FALSE)
```

	Model	AIC	BIC
1	gamma	1107.83	1131.70
2	weibull	1107.98	1131.84
3	negative-binomial	1109.15	1133.02
4	generalised-gamma	1108.72	1135.58
5	Poisson	2299.18	2320.07

Table 11: Information criteria results

According to Table 11, the gamma renewal model is slightly preferred to the Weibull model although since the maximum differences of AIC is less than 2 units, the three models can be roughly seen equivalent. The table also confirms that the Poisson model is not able to deal with the large over-dispersion.

We conclude this analysis by running a formal chi-square goodness of fit test (Cameron and Trivedi 2013, Section 5.3.4) to the results of the previously selected model.

```
R> gof <- chiSq_gof(gam, breaks = breaks_)
R> gof

chi-square goodness-of-fit test

Cells considered 0 1-2 3-4 5 6 7-8 9-11 12-14 15-16 17-22 23-26 27-31 >= 32
  DF Chisq Pr(>Chisq)
1 12  17.5       0.13
```

The null hypothesis cannot be rejected at standard confidence levels and we conclude that the selected model presents a good fit to the data. Users can check that the same test yields similar results for the Weibull and negative binomial models but comfortably rejects the null hypothesis for the Poisson and generalised gamma models. These results confirm what we observed before.

7. Conclusion and Future Work

Count regression models derived from renewal processes are a flexible class of models that extends the Poisson model and allows the use of inter-arrival times distributions that are more flexible than the exponential. The **Countr** package implements this class of models using standard R framework (very similar to `glm()`) and hence allows users familiar with the generalized linear models to experience a more flexible class of models with minimum additional effort. Usual methods for model fitting, goodness of fit diagnosis and prediction are also provided.

Countr currently contains four built-in distributions for which the computations are optimised by programming crucial parts in C++ and choosing taylored parameters for optimisation functions. Although users can define their own inter-arrival times distribution, this may result in long computation times as demonstrated in Section 5.3. In future versions of the package, a larger choice of survival distributions will be available.

Renewal regression models can be extended in many directions. One of them is to allow the time to the first event to have a different distribution from the inter-arrival times for later events. This gives rise to a type of hurdle model that we call "modified renewal processes". This family of models is being studied by the authors and an experimental version is shipped with **Countr**. Another direction in which the **Countr** package can be extended is by allowing multivariate (and especially bivariate) models to be fitted. A Copula (Cameron and Trivedi 2013, Section 8.5) can be used to take into account dependence between the count marginals. Such models will also be included in future versions of **Countr**.

8. Acknowledgments

We would like to thank two anonymous reviewers for their thorough comments and constructive suggestions, which helped us improve the quality of the software and the manuscript.

References

Aitkin M (1978). "The Analysis of Unbalanced Cross-Classifications." *Journal of the Royal Statistical Society A (General)*, pp. 195–223.

Baker R, Kharrat T (2017). "Event Count Distributions from Renewal Processes: Fast Computation of Probabilities." *IMA Journal of Management Mathematics*. URL http://dx.doi.org/10.1093/imaman/dpx008.

Bittner E, Nussbaumer A, Janke W, Weigel M (2007). "Self-Affirmation Model for Football Goal Distributions." *EPL (Europhysics Letters)*, **78**(5), 58002.

Boshnakov G, Kharrat T, McHale IG (2017). "A Bivariate Weibull Count Model for Forecasting Association Football Scores." *International Journal of Forecasting*, **33**(2), 458–466.

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data, 2nd ed.* Cambridge University Press.

Canty A, Ripley BD (2017). *boot: Bootstrap R (S-PLUS) Functions.* R package version 1.3-20.

Chambers JM, Hastie TJ (1992). *Statistical Models in S.* Wadsworth & Brooks/Cole, Pacific Grove, California.

Dahl DB (2016). *xtable: Export Tables to LaTeX or HTML.* R package version 1.8-2, URL https://CRAN.R-project.org/package=xtable.

Davison AC, Hinkley DV (1997). *Bootstrap Methods and Their Application*, volume 1. Cambridge University Press.

De Pril N (1985). "Recursions for Convolutions of Arithmetic Distributions." *Astin Bulletin*, **15**(02), 135–139.

Dixon M, Robinson M (1998). "A Birth Process Model for Association Football Matches." *Journal of the Royal Statistical Society D (The Statistician)*, **47**(3), 523–538.

Eddelbuettel D, Sanderson C (2014). "RcppArmadillo: Accelerating R With High-Performance C++ Linear Algebra." *Computational Statistics and Data Analysis*, **71**, 1054–1063. URL http://dx.doi.org/10.1016/j.csda.2013.02.005.

Feller W (1971). *An Introduction to Probability Theory and Its Applications; 2nd ed.* John Wiley & Sons, New York, NY.

Fox J, Weisberg S (2011). *An R Companion to Applied Regression.* Second edition. Sage, Thousand Oaks CA. URL http://socserv.socsci.mcmaster.ca/jfox/Books/Companion.

Jackson C (2016). "**flexsurv**: A Platform for Parametric Survival Modeling in R." *Journal of Statistical Software*, **70**(8), 1–33. doi:10.18637/jss.v070.i08.

Jose K, Abraham B (2013). "A Counting Process with Gumbel Inter-Arrival Times for Modeling Climate Data." *Journal of Environmental Statistics*, **4**(5). ISSN 1945-1296. URL http://jes.stat.ucla.edu/v04/i05.

Jose KK, Abraham B (2011). "A Count Model Based on Mittag-Leffler Inter-Arrival Times." *Statistica*, **71**(4), 501–514.

Kharrat T (2016). *A Journey Across Football Modelling with Application to Algorithmic Trading (PhD Thesis).* University of Manchester.

Kharrat T, Boshnakov GN (2016). ***Countr**: Flexible Univariate Count Models Based on Renewal Processes.* R package version 3.2.7, URL https://cran.r-project.org/web/packages/Countr/index.html.

Kharrat T, Boshnakov GN, McHale I, Baker R (2019). "Flexible Regression Models for Count Data Based on Renewal Processes: The Countr Package." *Journal of Statistical Software*, **90**(13), 1–35. doi:10.18637/jss.v090.i13.

McCullagh P, Nelder JA (1989). *Generalized Linear Models, 2nd ed., no. 37 in Monograph on Statistics and Applied Probability*. CRC press/Chapman & Hall.

McShane B, Adrian M, Bradlow ET, Fader PS (2008). "Count Models Based on Weibull Inter-Arrival Times." *Journal of Business & Economic Statistics*, **26**(3).

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: **optimx** for R." *Journal of Statistical Software*, **43**(9), 1–14. URL http://www.jstatsoft.org/v43/i09/.

Prentice RL (1974). "A Log Gamma Model and Its Maximum Likelihood Estimation." *Biometrika*, **61**(3), 539–544.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007). *Numerical Recipes: The Art of Scientific Computing (3rd ed.)*. Cambridge University Press.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Stacy EW (1962). "A Generalization of the Gamma Distribution." *The Annals of Mathematical Statistics*, pp. 1187–1192.

Tadikamalla PR (1980). "A Look at the Burr and Related Distributions." *International Statistical Review/Revue Internationale de Statistique*, pp. 337–344.

Varadhan R, Gilbert P (2009). "**BB**: An R Package for Solving a Large System of Nonlinear Equations and for Optimizing a High-Dimensional Nonlinear Objective Function." *Journal of Statistical Software*, **32**(4), 1–26. URL http://www.jstatsoft.org/v32/i04/.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer-Verlag, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wickham H, Francois R (2016). *dplyr: A Grammar of Data Manipulation*. R package version 0.5.0, URL https://CRAN.R-project.org/package=dplyr.

Winkelmann R (1995). "Duration Dependence and Dispersion in Count-Data Models." *Journal of Business & Economic Statistics*, **13**(4), 467–474.

Winkelmann R (2013). *Econometric Analysis of Count Data, 4th ed.* Springer-Verlag, Berlin Heidelberg.

Zeileis A, Kleiber C, Jackman S (2008). "Regression Models for Count Data in R." *Journal of statistical software*, **27**(8), 1–25.

Affiliation:

Tarak Kharrat
School of Marketing and Operations
University of Liverpool
33 Finsbury Square, London EC2A 1AG, United Kingdom
E-mail: Tarak.Kharrat@liverpool.ac.uk

Georgi N. Boshnakov
Department of Mathematics
The University of Manchester
Oxford Road, Manchester M13 9PL, UK
URL: https://personalpages.manchester.ac.uk/staff/georgi.boshnakov/