Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. doi: 10.18637/jss.v000.i00

singleRcapture: A Package for Single-Source Capture-Recapture Models

Piotr Chlebicki
Stockholm University

Maciej Beręsewicz
Poznań University of Economics and Business
Statistical Office in Poznań

Abstract

Estimating population size is an important issue in official statistics, social sciences and natural sciences. One way to approach this problem is to use capture-recapture methods, which can be classified according to the number of sources used, the most important distinction from the prespective of this work being between methods based on one source and those based on two or more sources. In this presentation we will introduce the **singleRcapture** R package for fitting SSCR models. The package implements state-of-the-art models as well as some new models proposed by the authors (e.g. extensions of zero-truncated one-inflated and one-inflated zero-truncated models). The software is intended for users interested in estimating the size of populations, particularly those that are difficult to reach or for which information is available from only one source and dual/multiple system estimation cannot be used.

Keywords: population size estimation, hidden populations, truncated distributuons, count regression models, R.

1. Introduction

Population size estimation is a methodological approach employed across multiple scientific disciplines, serving as a basis for research, policy formulation, and decision-making processes (Böhning, Bunge, and Heijden 2018). In the field of statistics, particularly official statistics, precise population estimates are essential for developing robust economic models, optimizing resource allocation, and informing evidence-based policy formulation (cf. Baffour-Awuah 2009). Social scientists utilize advanced population estimation techniques to investigate *hard-to-reach* populations, such as homeless individuals or illicit drug users, thereby addressing the inherent limitations of conventional census methodologies. These techniques are crucial for obtaining accurate data on populations that are typically under-represented or difficult to access through traditional sampling methods (Vincent and Thompson 2022). In ecology and epidemiology, researchers focus on estimating the size of specific species or disease-affected populations within defined geographical regions, which is vital for conservation efforts, ecosystem management, and public health interventions.

Population size estimation can be approached through various methodologies, each with distinct advantages and limitations. Traditional approaches include full enumeration (e.g. census operations) and comprehensive sample surveys, which, while providing detailed data, are often resource-intensive and may result in delayed estimates, particularly for human populations. Alternative methods leverage existing data sources, such as administrative registers or carefully designed small-scale studies in wildlife research or census coverage surveys (Wolter 1986; Zhang 2019). Application of these sources often comes with statistical methods, known as *capture-recapture* or *multiple system estimation*, that utilizes data from multiple enumerations of the same population (cf. Dunne and Zhang 2024). This can be implemented using a single source with repeated observations, two, or multiple sources.

In this paper we focus methods that utilize a single data source with multiple enumerations of the same units (cf. van der Heijden, Bustami, Cruyff, Engbersen, and van Houwelingen 2003). In human population studies, such data might be derived from police records, health system databases, or border control logs, while for non-human populations, veterinary records or specialized field data serve as analogous sources. These methods are often applied for hard-to-reach or hidden population where standard sampling methods may be inappropriate because of the costs or problems with identification of members of these populations.

While methods for two or more sources are implemented in various open-source software (e.g., Baillargeon and Rivest 2007) the single-source capture-recapture (SSCR) methods are less available being only partially implemented in existing R packages. The goal of the paper is to introduce the **singleRcapture** and **singleRcaptureExtra** packages which by implementing state-of-the-art methods in SSCR and providing user friendly API which mimics existing R functions (e.g., `glm`) attemp to bridge this aforementioned gap. In the next subsection we descirbe the available R packages that could be used for estimating population size based on SSCR methods.

1.1. Software for capture-recapture for single and multiple sources

Majority of SSCR methods assume zero-truncated distributions or their extensions (e.g., inclusion of one-inflation). The **countreg** (Zeileis, Kleiber, and Jackman 2008), **VGAM** (Yee 2015) or **distributions3** (Hayes, Moller-Trane, Jordan, Northrop, Lang, and Zeileis 2024) implement some of those truncated distributions and the most general distributions such as Generally Altered, Inflated, Truncated and Deflated (GAITD) can be found in the **VGAM**. However, estimation of parameters of a given truncated (and possibly inflated) distribution is just a first step (similarly as in log-linear models in capture-recapture with two sources) and to best of our knowledge there is no open-source software that allows to estimate population size based on SSCR method, including variance estimator or diagnostics.

Therefore, the goal of the **singleRcapture** is R language is to bridge this gap to provide scientists and other practitioners a tool for estimation of population size based on SSCR methods. The package implements state-of-the-art methods as recently described by Böhning *et al.* (2018) or Böhning and Friedl (2024) and its extensions (e.g., inclusion of covariates, different treatment of one-inflation) that we will cover in detail in Section 1. The package implements variance estimation based on various methods, allows for implementing custom models as well as diagnostics plots (e.g. rootograms) with parameters estimated using a modified IRLS algorithm implemented by us to for estimation stability. Furthermore, as many R users are familiar with **countreg** or **VGAM** we have implemented a lightweight extension **singleRcaptureExtra**, available through Github (`https://github.com/ncn-foreigners/singleRcaptureExtra`), that allows for integration of **singleRcapture** with those packages.

The remaining part of the paper is as follows. In Section 2 a brief description of the theoretical background is given and information on the fitting methods, the available methods and variance estimation is presented. In Section 3 the main functionalities of the package are introduced, and the main `S3` methods as well as implemented diagnostics and useful functions are covered. Section 4 covers integration with **countreg** and **VGAM** packages through **singleRcaptureExtra** package. The paper ends with conclusions and an appendix that shows how to a implement custom model and how one can use the `estimatePopsizeFit` which is faster than the main function but only estimates regression, which could be of interest to users interested in using any new bootstrap methods not programmed in the package.

2. Theoretical background

2.1. How do we estimate population size with a single register?

Let Y_k represent the number of times k-th unit was observed in a register. Clearly, we only observe $k : Y_k > 0$ and we do not know how many units are missed (i.e. $Y_k = 0$) and to find the population size denoted by N we need to estimate it. In general, we assume that conditional distribution of Y_k given a vector of covariates $\boldsymbol{x}_k$ follows some version of zero-truncated count data distribution (and its extensions). Knowing the parameters of the distribution we may estimate the population size using Horowitz-Thompson type estimator given by:

$$\hat{N} = \sum_{k=1}^{N} \frac{I_k}{\mathbb{P}[Y_k > 0|\boldsymbol{X}_k]} = \sum_{k=1}^{N_{obs}} \frac{1}{\mathbb{P}[Y_k > 0|\boldsymbol{X}_k]}, \tag{1}$$

where $I_k := \mathcal{I}_{\mathbb{N}}(Y_k)$, and maximum likelihood estimate of N is obtained after substituting regression estimates for $\mathbb{P}[Y_k > 0|\boldsymbol{x}_k]$ into (1).

The basic SSCR assumes independence between counts which may be rather naive as the first capture may significantly influence the behaviour of a given unit or limit possibilities of further captures (e.g. due to incarceration). To solve these issues, Godwin and Böhning (2017a) and Godwin and Böhning (2017b) introduced one-inflated distributions that explicitly model probability of the singletons by giving additional mass ω for singletons denoted as $\mathcal{I}_{\{1\}}(y)$:

$$\mathbb{P}^*[Y = y|Y > 0] = \omega\mathcal{I}_{\{1\}}(y) + (1 - \omega)\mathbb{P}[Y = y|Y > 0].$$

For more about the one-inflation in the context SSCR see recent review of Böhning and Friedl (2024).

The analytic variance estimation is then done by computing two parts of the decomposition due to the law of total variance given by:

$$\text{var}[\hat{N}] = \mathbb{E}\left[\text{var}\left[\hat{N}|I_1,\ldots,I_n\right]\right] + \text{var}\left[\mathbb{E}[\hat{N}|I_1,\ldots,I_n]\right], \tag{2}$$

where the first part can be estimated using the multivariate δ method given by:

$$\mathbb{E}\left[\text{var}\left[\hat{N}|I_1,\ldots,I_n\right]\right] = \left(\frac{\partial(N|I_1,\ldots,I_N)}{\partial\boldsymbol{\beta}}\right)^\top \text{cov}\left[\hat{\boldsymbol{\beta}}\right]\left(\frac{\partial(N|I_1,\ldots,I_N)}{\partial\boldsymbol{\beta}}\right)\bigg|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}},$$

while the second part of the decomposition in (2) is under the assumption of independence of I_k's and after some omitted simplifications one sees that this is optimally estimated by:

$$\text{var}\left(\mathbb{E}(\hat{N}|I_1,\ldots,I_n)\right) = \text{var}\left(\sum_{k=1}^{N}\frac{I_k}{\mathbb{P}(Y_k>0)}\right) \approx \sum_{k=1}^{N_{obs}}\frac{1-\mathbb{P}(Y_k>0)}{\mathbb{P}(Y_k>0)^2},$$

which forms the basis for the interval estimation. Confidence intervals are usually constructed under the assumption of (asymptotic) normality of $\hat{N}$ or asymptotic normality of $\ln(\hat{N}-N)$ (or log normality of $\hat{N}$). The latter of which is an attempt to address a common criticism of student type confidence intervals in SSCR, that is a possibly skewed distribution of $\hat{N}$, and results in the $1-\alpha$ confidence interval given by:

$$\left(N_{obs} + \frac{\hat{N}-N_{obs}}{\xi}, N_{obs} + \left(\hat{N}-N_{obs}\right)\xi\right),$$

where:

$$\xi = \exp\left(z\left(1-\frac{\alpha}{2}\right)\sqrt{\ln\left(1+\frac{\widehat{\text{Var}}(\hat{N})}{\left(\hat{N}-N_{obs}\right)^2}\right)}\right).$$

and where z is the quatile function of the standard normal distribution. The estimator $\hat{N}$ is best interpreted as being an estimator for the total number of observable units in the population since we have no means of estimating the number of units in the population for which the probability of being included in the data is 0 (cf. van der Heijden *et al.* 2003).

2.2. Available models

The full list of implemented models in **singleRcapture** along with the expressions for probability density functions and point estimates can be found in the collective help file for all family functions:

```
R> ?ztpoisson
```

For the sake of simplicity we limit ourselves to just listing the family functions:

- Generalized Chao's (Chao 1987) and Zelterman's (Zelterman 1988) estimators via logistic regression on variable Z defined as $Z = 1$ if $Y = 2$ and $Z = 0$ if $Y = 1$ with

$Z \sim b(p)$ where $b(\cdot)$ is the Bernoulli distribution and p can be modeled for each unit k by $\text{logit}(p_k) = \ln(\lambda_k/2)$ with Poisson parameter $\lambda_k = \boldsymbol{x}_k\boldsymbol{\beta}$ (for covariate extension see Böhning, Vidal-Diez, Lerdsuwansri, Viwatwongkasem, and Arnold (2013) and Böhning and van der Heijden (2009)):

$$\hat{N} = N_{obs} + \sum_{k=1}^{\boldsymbol{f}_1+\boldsymbol{f}_2} \left(2\exp\left(\boldsymbol{x}_k\hat{\boldsymbol{\beta}}\right) + 2\exp\left(2\boldsymbol{x}_k\hat{\boldsymbol{\beta}}\right)\right)^{-1}, \qquad \text{(Chao's estimator)}$$

$$\hat{N} = \sum_{k=1}^{N_{obs}} \left(1 - \exp\left(-2\exp\left(\boldsymbol{x}_k\hat{\boldsymbol{\beta}}\right)\right)\right)^{-1}. \qquad \text{(Zelterman's estimator)}$$

- Zero-truncated (`zt`*) and zero-one-truncated (`ztoi`*) Poisson (cf. Böhning and van der Heijden 2019), geometric, NB type II (NB2) regression where the non-truncated distribution is parameterized as:

$$\mathbb{P}[Y = y|\lambda, \alpha] = \frac{\Gamma\left(y + \alpha^{-1}\right)}{\Gamma\left(\alpha^{-1}\right) y!} \left(\frac{\alpha^{-1}}{\alpha^{-1} + \lambda}\right)^{\alpha^{-1}} \left(\frac{\lambda}{\lambda + \alpha^{-1}}\right)^{y}.$$

- Zero-truncated one-inflated (`ztoi`*) modifications distributions where the new probability $\mathbb{P}^*$ measure is defined in terms of count data measure $\mathbb{P}$ with support on $\mathbb{N} \cup \{0\}$ as:

$$\mathbb{P}^*[Y = y] = \begin{cases} \mathbb{P}[Y = 0] & y = 0, \\ \omega\left(1 - \mathbb{P}[Y = 0]\right) + (1 - \omega)\mathbb{P}[Y = 1] & y = 1, \\ (1 - \omega)\mathbb{P}[Y = y] & y > 1, \end{cases}$$

$$\mathbb{P}^*[Y = y|Y > 0] = \omega\mathcal{I}_{\{1\}}(y) + (1 - \omega)\mathbb{P}[Y = y|Y > 0].$$

- One-inflated zero-truncated (`oizt`*) modifications distributions where the new probability $\mathbb{P}^*$ measure is defined as:

$$\mathbb{P}^*[Y = y] = \omega\mathcal{I}_{\{1\}}(y) + (1 - \omega)\mathbb{P}[Y = y],$$

$$\mathbb{P}^*[Y = y|Y > 0] = \omega\frac{\mathcal{I}_{\{1\}}(y)}{1 - (1 - \omega)\mathbb{P}[Y = 0]} + (1 - \omega)\frac{\mathbb{P}[Y = y]}{1 - (1 - \omega)\mathbb{P}[Y = 0]}.$$

  Note that `ztoi`* and `oizt`* distributions are equivalent as shown by Böhning (2023) but population size estimators are different.

In addition, we have provided two new approaches that allow modelling singletons in a similar was as in Hurdle models. In particular we have proposed the following:

- Zero-truncated Hurdle model (`ztHurdle`*) for Poisson, geometric and NB2 is defined as:

$$\mathbb{P}^*[Y = y] = \begin{cases} \frac{\mathbb{P}[Y=0]}{1-\mathbb{P}[Y=1]} & y = 0, \\ \pi(1 - \mathbb{P}[Y = 1]) & y = 1, \\ (1 - \pi)\frac{\mathbb{P}[Y=y]}{1-\mathbb{P}[Y=1]} & y > 1, \end{cases}$$

$$\mathbb{P}^*[Y = y|Y > 0] = \pi\mathcal{I}_{\{1\}}(y) + (1 - \pi)\mathcal{I}_{\mathbb{N}\setminus\{1\}}(y)\frac{\mathbb{P}[Y = y]}{1 - \mathbb{P}[Y = 0] - \mathbb{P}[Y = 1]}.$$

- The Hurdle zero-truncated (`Hurdlezt*`) for Poisson, geometric and NB2 is defined as:

$$\mathbb{P}^*[Y=y] = \begin{cases} \pi & y=1, \\ (1-\pi)\frac{\mathbb{P}[Y=y]}{1-\mathbb{P}[Y=1]} & y\neq 1, \end{cases}$$

$$\mathbb{P}^*[Y=y|Y>0] = \begin{cases} \pi\frac{1-\mathbb{P}[Y=1]}{1-\mathbb{P}[Y=0]-\mathbb{P}[Y=1]} & y=1, \\ (1-\pi)\frac{\mathbb{P}[Y=y]}{1-\mathbb{P}[Y=0]-\mathbb{P}[Y=1]} & y>1. \end{cases}$$

The approaches presented above differ in terms of assumptions, computational complexity, or how they treat heterogeneity of captures and singletons. For instance, the dispersion parameter α in the NB2 type models is often interpreted as measuring the *severeness* of unobserved heterogeneity in the underlying poisson process (cf. Cruyff and van der Heijden 2008). When using any truncated NB model the hope is that due to the class of models considered the consistency is not lost despite the lack of information.

While not discussed in the literature yet the interpretation of heterogeneous α across the population (specified in `controlModel`) would be that the unobserved heterogeneity affects the accuracy of the prediction for the dependent variable Y more severely than others. The geometric model (NB with $\alpha = 1$) is singled out in the package and often considered in the literature due to inherent computational issues with NB models which are exasperated by the fact that data in SSCR is usually of somewhat low quality. Sparseness of the data is in particular a common issue in SSCR and a big issue for all numerical methods for fitting the (zero-truncated) NB model.

The extra mass ω in the one-inflated models is an important extension to the researcher's toolbox for SSCR models. Since the inflation at $y = 1$ is likely to occur in many types of applications. For example in estimating the number active people who committed criminal acts in a given time period being observed naturally induces a risk of no longer being able to be observed for all units with possibility of arrest. One constraint present in modelling via inflated models is that trying to include both the possibility of one inflation and one deflation leads to both numerical and theoretical problems since the parameter space (of (ω, λ) or $(\omega, \lambda, \alpha)$) is then a much more complicated set.

Hurdle models are another approach to modelling the one-inflation, they can also model deflation as well as both inflation and deflation simultaneously so they are more flexible and situationally the Hurdle zero-truncated models seem to be more numerically stable.

Although interpretation of regression parameters tends to be somewhat overlooked in the SSCR studies we should point out that interpretation of the ω inflation parameter (in `ztoi*` or `oizt*`) is more convenient that the interpretation of the π probability parameter (in Hurdle models). Additionally the interpretation of the λ parameter in (one) inflated models conforms to the intuition that given that unit k comes from the non-inflated part of the population then it follows a poisson distribution (respectively geometric or negative binomial) with the λ parameter (or λ, α), in hurdle models one loses that interpretation. It is somewhat interesting is that the estimates from Hurdle zero-truncated and one-inflated zero-truncated models are "usually" quite close to one another, this however require more studies.

2.3. Fitting method

As previously noted the **singleRcapture** package supports modelling (linear) dependence on covariates of all parameters. To that end a modified IRLS algorithm is employed, full details are available in Yee (2015). In order to employ the algorithm a modified model matrix is created $\boldsymbol{X}_{\text{vlm}}$ at call to `estimatePopsize`. In the context of the models implemented in **singleRcapture** this matrix can be written as:

$$\boldsymbol{X}_{vlm} = \begin{pmatrix} \boldsymbol{X}_1 & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{X}_2 & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{X}_p \end{pmatrix} \tag{3}$$

where each $\boldsymbol{X}_i$ corresponds to a model matrix associated with user specified formula.

In the context of multi-parameter families we have a matrix of linear predictors $\boldsymbol{\eta}$ instead of a vector, with the number of columns matching the number of parameters in the distribution. "Weights" are then modified to be information matrices $\mathbb{E}\left[-\frac{\partial^2 \ell}{\partial \boldsymbol{\eta}_{(k)}^\top \partial \boldsymbol{\eta}_{(k)}}\right]$ where $\boldsymbol{\eta}_{(k)}$ is the k'th row of $\boldsymbol{\eta}$, while in the usual IRLS they are scalars $\mathbb{E}\left[-\frac{\partial^2 \ell}{\partial \eta_k^2}\right]$ which is often just $-\frac{\partial^2 \ell}{\partial \eta^2}$.

Algorithm 1: A modified IRLS algorithm used in the **singleRcapture** package

1 Initialize with `iter`$\leftarrow 1, \boldsymbol{\eta} \leftarrow$`start` $, \boldsymbol{W} \leftarrow I, \ell \leftarrow \ell(\boldsymbol{\beta})$.

2 Store values from the previous step: $\ell_{-} \leftarrow \ell, \boldsymbol{W}_{-} \leftarrow \boldsymbol{W}, \boldsymbol{\beta}_{-} \leftarrow \boldsymbol{\beta}$ (the last assignment is omitted during the first iteration), and assign values in current iteration
$\boldsymbol{\eta} \leftarrow \boldsymbol{X}_{\text{vlm}}\boldsymbol{\beta} + \boldsymbol{o}, \boldsymbol{W}_{(k)} \leftarrow \mathbb{E}\left[-\frac{\partial^2 \ell}{\partial \boldsymbol{\eta}_{(k)}^\top \partial \boldsymbol{\eta}_{(k)}}\right], Z \leftarrow \boldsymbol{\eta}_{(k)} + \frac{\partial \ell}{\partial \boldsymbol{\eta}_{(k)}} \boldsymbol{W}_{(k)}^{-1} - \boldsymbol{o}_{(k)}$.

3 Assign current coefficient value: $\boldsymbol{\beta} \leftarrow \left(\boldsymbol{X}_{\text{vlm}} \boldsymbol{W} \boldsymbol{X}_{\text{vlm}}\right)^{-1} \boldsymbol{X}_{\text{vlm}} \boldsymbol{W} \boldsymbol{Z}$.

4 If $\ell(\boldsymbol{\beta}) < \ell(\boldsymbol{\beta}_{-})$ try selecting the smallest value h such that for $\boldsymbol{\beta}_h \leftarrow 2^{-h}\left(\boldsymbol{\beta} + \boldsymbol{\beta}_{-}\right)$ the inequality $\ell(\boldsymbol{\beta}_h) > \ell(\boldsymbol{\beta}_{-})$ holds if this is successful $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_h$ else stop the algorithm.

5 If convergence is achieved or `iter` is higher than `maxiter` end algorithm, else `iter`$\leftarrow 1+$`iter` and return to step 2.

2.4. Variance estimation using bootstrap

We have implemented three types of bootstrap algorithms: parametric, semi-parametric and nonparametric with the nonparametric being bootstrap being the usual bootstrap algorithm which as argued in Norris and Pollock (1996) and Zwane and Van der Heijden (2003).

The idea of semi-parametric bootstrap is to modify the usual bootstrap to include the additional uncertainty due to the sample size being a random variable. This type of bootstrap can be in short described as in the Algorithm 2.

Algorithm 2: Semi-parametric bootstrap

1 Draw the sample size $N_{obs}' \sim \text{Be}\left(N', \frac{N_{obs}}{N'}\right)$, where $N' = \lfloor \hat{N} \rfloor + b\left(\lfloor \hat{N} \rfloor - \hat{N}\right)$.

2 Draw N_{obs}' units from the data uniformly without replacement.

3 Obtain new population size estimate N_b using bootstrap data.

4 Repeat $1-3$ B times.

In other words, we first draw the sample size and then the sample conditional on the sample size. Note that in using semi-parametric bootstrap one implicitly assumes that the population size estimate $\hat{N}$ is accurate. The last implemented bootstrap type is the parametric algorithm which in short first draws the finite population of size $\approx \hat{N}$ from the superpopulation model and then samples from this population according to the selected model as described in Algorithm 3.

Algorithm 3: Parametric bootstrap

1 Draw the number of covariates equal to $\lfloor \hat{N} \rfloor + b\left(\lfloor \hat{N} \rfloor - \hat{N}\right)$ proportional to the estimated contribution $(\mathbb{P}\left[Y_k > 0 | \boldsymbol{x}_k\right])^{-1}$ with replacement.
2 Using the fitted model and regression coefficients $\hat{\boldsymbol{\beta}}$ draw for each covariate the Y value from the corresponding probability measure on $\mathbb{N} \cup \{0\}$.
3 Truncate units with drawn Y value equal to 0.
4 Obtain population size estimate N_b based on the truncated data.
5 Repeat $1 - 3$ B times.

Note that for this type of algorithm to result in consistent standard error estimates it is imperative that the estimated model for the entire superpopulation probability space is consistent which may be much less realistic than semi-parametric bootstrap. The parametric bootstrap algorithm is the default in **singleRcapture**.

3. Basic usage

3.1. The main function

The main function that **singleRcapture** is built around is `estimatePopsize`. The leading design principle was to make using `estimatePopsize` as close to standard `stats::glm` as possible. The most important arguments are:

- `formula` – the main formula (i.e for the Poisson λ parameter),
- `data` – the `data.frame` (or `data.frame` coercible) object,
- `model` – either a function a string or a family class object specifying which model should be used possible values are listed in documentation. The supplied argument should have the form `model = "ztpoisson"`, `model = ztpoisson` or if link function should be specified then `model = ztpoisson(lambdaLink = "log")` can be used,
- `method` – numerical method used to fit regression `IRLS` or `optim`,
- `popVar` – a method for estimating variance of $\hat{N}$ and confidence interval creation (either bootstrap, analytic or skipping the estimation entirely),
- `controlMethod, controlModel, controlPopVar` – control parameters for numerical fitting, specifying additional formulas (inflation, dispersion) and population size estimation respectively,

- `offset` – a matrix of offset values with number of columns matching the number of distribution parameters providing offset values to each of linear predictors.

With the `formula, data, model` being the three arguments which must be provided in `estimatePopsize` syntax.

3.2. Example with R code

The package can be installed in a standard way using:

```
R> install.packages("singleRcapture")
```

To show the main function let us recreate the zero-truncated Poisson model from van der Heijden *et al.* (2003) on the same data included in the package under the name `netherlandsimmigrant`:

```
R> library(singleRcapture)
R> knitr::kable(head(netherlandsimmigrant))
```

capture	gender	age	reason	nation
1	male	<40yrs	Other reason	North Africa
1	male	<40yrs	Other reason	North Africa
1	male	<40yrs	Other reason	North Africa
1	male	<40yrs	Other reason	Asia
1	male	<40yrs	Other reason	Asia
2	male	<40yrs	Other reason	North Africa

This data set contains information about immigrants in four cities (Amsterdam, Rotterdam, The Hague and Utrecht) in Netherlands that have been staying in the country without a legal permit in 1995 and have appeared in police records that year. The number of times each individual appeared in the records is included in the `capture` variable with the available covariates being `gender, age, reason, nation` being respectively the persons gender and age, reason for being captured and region of the world from which each person comes:

```
R> summary(netherlandsimmigrant)
```

```
    capture          gender          age                 reason    
 Min.   :1.000   female: 398   <40yrs:1769   Illegal stay: 259  
 1st Qu.:1.000   male  :1482   >40yrs: 111   Other reason:1621  
 Median :1.000                                                  
 Mean   :1.162                                                  
 3rd Qu.:1.000                                                  
 Max.   :6.000                                                  
                   nation    
 American and Australia: 173  
 Asia                  : 284  
```

```
 North Africa          :1023
 Rest of Africa        : 243
 Surinam               :  64
 Turkey                :  93
```

One point which we should make while analysing this data set is that there is a disproportionate number of individuals who were observed only once:

```
R> knitr::kable(t(table(netherlandsimmigrant$capture)))
```

1	2	3	4	5	6
1645	183	37	13	1	1

The basic syntax is vary similar to that of `glm` with the output of the summary method being also quite similar except for the additional results of the population size estimates:

```
R> basicModel <- estimatePopsize(
+   formula = capture ~ gender + age + nation,
+   model   = ztpoisson(),
+   data    = netherlandsimmigrant
+ )
R> summary(basicModel)


Call:
estimatePopsize.default(formula = capture ~ gender + age + nation,
    data = netherlandsimmigrant, model = ztpoisson())

Pearson Residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.486442 -0.486442 -0.298080  0.002093 -0.209444 13.910844

Coefficients:
-----------------------
For linear predictors associated with: lambda
                     Estimate Std. Error z value  P(>|z|)
(Intercept)           -1.3411     0.2149  -6.241 4.35e-10 ***
gendermale             0.3972     0.1630   2.436 0.014832 *
age>40yrs             -0.9746     0.4082  -2.387 0.016972 *
nationAsia            -1.0926     0.3016  -3.622 0.000292 ***
nationNorth Africa     0.1900     0.1940   0.979 0.327398
nationRest of Africa  -0.9106     0.3008  -3.027 0.002468 **
nationSurinam         -2.3364     1.0136  -2.305 0.021159 *
nationTurkey          -1.6754     0.6028  -2.779 0.005445 **
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

AIC: 1712.901
BIC: 1757.213
Residual deviance: 1128.553

Log-likelihood: -848.4504 on 1872 Degrees of freedom
Number of iterations: 8
-----------------------
Population size estimation results:
Point estimate 12690.35
Observed proportion: 14.8% (N obs = 1880)
Std. Error 2808.169
95% CI for the population size:
          lowerBound upperBound
normal      7186.444   18194.26
logNormal   8431.275   19718.32
95% CI for the share of observed population:
          lowerBound upperBound
normal     10.332927   26.16037
logNormal   9.534281   22.29793
```

According to this simple model the population size is about 12.5k with about 15% of units observed in the register. The 95% CI under normality indicate that the true population size may be between 7k-18k with about 10% to 26% observed in the register.

Since there is a reasonable suspicion that the act of observing a unit in the dataset may led to undesirable consequences from the point of view of the subject of the observation (here possible deportation, detainment or similar). For those reason researcher may consider one-inflated models such as `oiztgeom` and presented below.

```
R> set.seed(123456)
R> modelInflated <- estimatePopsize(
+     formula = capture ~ nation,
+     model   = oiztgeom(omegaLink = "cloglog"),
+     data    = netherlandsimmigrant,
+     controlModel = controlModel(
+         omegaFormula = ~ gender + age
+     ),
+     popVar = "bootstrap",
+     controlPopVar = controlPopVar(bootType = "semiparametric")
+ )
R> summary(modelInflated)


Call:
estimatePopsize.default(formula = capture ~ nation, data = netherlandsimmigrant,
```

```
    model = oiztgeom(omegaLink = "cloglog"), popVar = "bootstrap",
    controlModel = controlModel(omegaFormula = ~gender + age),
    controlPopVar = controlPopVar(bootType = "semiparametric"))

Pearson Residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.41643 -0.41643 -0.30127  0.00314 -0.18323 13.88376

Coefficients:
-----------------------
For linear predictors associated with: lambda
                     Estimate Std. Error z value  P(>|z|)
(Intercept)           -1.2552     0.2149  -5.840 5.22e-09 ***
nationAsia            -0.8193     0.2544  -3.220  0.00128 **
nationNorth Africa     0.2057     0.1838   1.119  0.26309
nationRest of Africa  -0.6692     0.2548  -2.627  0.00862 **
nationSurinam         -1.5205     0.6271  -2.425  0.01532 *
nationTurkey          -1.1888     0.4343  -2.737  0.00619 **
-----------------------
For linear predictors associated with: omega
            Estimate Std. Error z value  P(>|z|)
(Intercept)  -1.4577     0.3884  -3.753 0.000175 ***
gendermale   -0.8738     0.3602  -2.426 0.015267 *
age>40yrs     1.1745     0.5423   2.166 0.030326 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

AIC: 1677.125
BIC: 1726.976
Residual deviance: 941.5416

Log-likelihood: -829.5625 on 3751 Degrees of freedom
Number of iterations: 10
-----------------------
Population size estimation results:
Point estimate 6699.953
Observed proportion: 28.1% (N obs = 1880)
Boostrap sample skewness: 1.621389
0 skewness is expected for normally distributed variable
---
Bootstrap Std. Error 1719.353
95% CI for the population size:
lowerBound upperBound
  5001.409  11415.969
95% CI for the share of observed population:
lowerBound upperBound
  16.46816   37.58941
```

This approach suggest that the population size is about 7k which is about 5k less than the naive Poisson approach. Comparison of AIC and BIC suggest that the one-inflation model fits the data better with BIC for `oiztgeom` 1727 and 1757 for `ztpoisson`.

3.3. Methods

For the purpose of the package we have created a class `singleRStaticCountData`, `singleR popSizeEstResults` which allows for extracting relevant information regarding the population size. For instance, function `popSizeEst` allows to extract information on the estimated size of the population as given below:

```
R> (popEst <- popSizeEst(basicModel))

Point estimate: 12690.35
Variance: 7885812
95% confidence intervals:
          lowerBound upperBound
normal      7186.444   18194.26
logNormal   8431.275   19718.32
```

and the resulting object `popEst` is of the `popSizeEstResults` class contains the following fields:

- `pointEstimate`, `variance` – numerics containing point estimate and variance of this estimate.
- `confidenceInterval` – a `data.frame` with confidence intervals.
- `boot` – If bootstrap was performed a numeric vector containing the $\hat{N}$ values from the bootstrap, a character vector with value `"No bootstrap performed"` otherwise.
- `control` – a `controlPopVar` object with controls used to obtained the object.

3.4. Testing marginal frequencies

A popular method of testing the model fit in single source capture-recapture studies is comparing the fitted marginal frequencies $\sum_{j=1}^{N_{obs}} \hat{\mathbb{P}}\left[Y_j = k|\boldsymbol{x}_j, Y_j > 0\right]$ with the observed marginal frequencies $\sum_{j=1}^{N} \mathcal{I}_{\{k\}}(Y_k) = \sum_{j=1}^{N_{obs}} \mathcal{I}_{\{k\}}(Y_k)$ for $k \geq 1$. If a fitted model bears sufficient resemblance to the real data collection process these quantities should be quite close and both G and χ^2 tests may be employed in order to test the statistical significance of the discrepancy with the following **singleRcapture** syntax for the Poisson model (rather poor fit):

```
R> margFreq <- marginalFreq(basicModel)
R> summary(margFreq, df = 1, dropl5 = "group")
```

```
Test for Goodness of fit of a regression model:

                 Test statistics df P(>X^2)
Chi-squared test           50.06  1 1.5e-12
G-test                     34.31  1 4.7e-09

--------------------------------------------------------------
Cells with fitted frequencies of < 5 have been grouped
Names of cells used in calculating test(s) statistic: 1 2 3
```

and for the one-inflated model (better fit):

```
R> margFreq_inf <- marginalFreq(modelInflated)
R> summary(margFreq_inf, df = 1, dropl5 = "group")
```

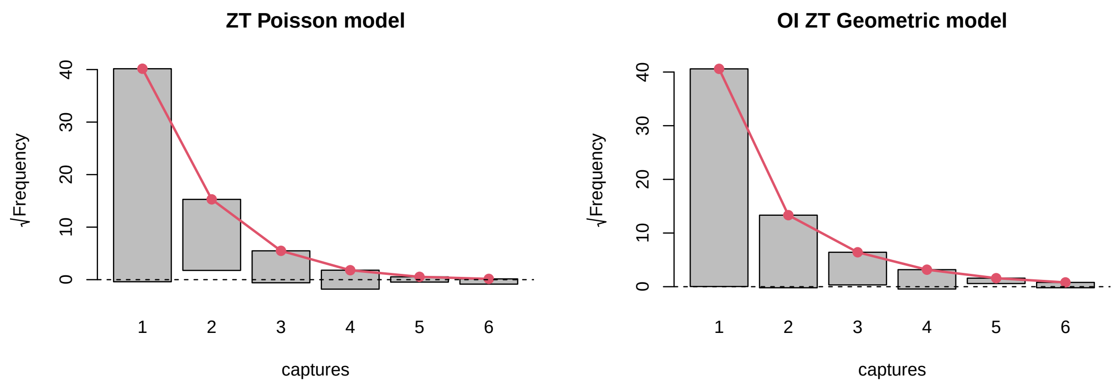
```
Test for Goodness of fit of a regression model:

                 Test statistics df P(>X^2)
Chi-squared test            1.88  1    0.17
G-test                      2.32  1    0.13

--------------------------------------------------------------
Cells with fitted frequencies of < 5 have been grouped
Names of cells used in calculating test(s) statistic: 1 2 3 4
```

where the `dropl5` argument is used to indicate how to handle the cells with less than 5 fitted observations, note however that currently there is no continuity correction.

3.5. Diagnostics

The `singleRStaticCountData` class has a `plot` method implementing several types of quick demonstrative plots such as the rootogram (cf. Kleiber and Zeileis 2016) for comparing the fitted and marginal frequencies which we can get with the syntax:

```
R> plot(   basicModel, plotType = "rootogram", main = "ZT Poisson model")
R> plot(modelInflated, plotType = "rootogram", main = "OI ZT Geometric model")
```



Plots suggest that the `otztgeom` model fits the data better. Furthermore, important issue in population size estimation is the diagnostics of the models in order to verify whether influential observations are present in the data. For this purpose leave-one-out (LOO) diagnostic implemented in the `dfbeta` from the **stats** package was adapted and demonstrated below:

```
R> dfb <- dfbeta(basicModel)
R> knitr::kable(tibble::as_tibble(t(apply(dfb, 2, quantile)*100)), digits = 4)
```

0%	25%	50%	75%	100%
-0.9909	-0.1533	0.0191	0.0521	8.6619
-9.0535	-0.0777	-0.0283	0.1017	2.2135
-2.0010	0.0179	0.0379	0.0691	16.0061
-9.5559	-0.0529	0.0066	0.0120	17.9914
-9.6605	-0.0842	-0.0177	0.0087	3.1260
-9.4497	-0.0244	0.0030	0.0083	10.9787
-9.3140	-0.0066	0.0020	0.0035	99.3383
-9.6198	-0.0220	0.0079	0.0143	32.0980

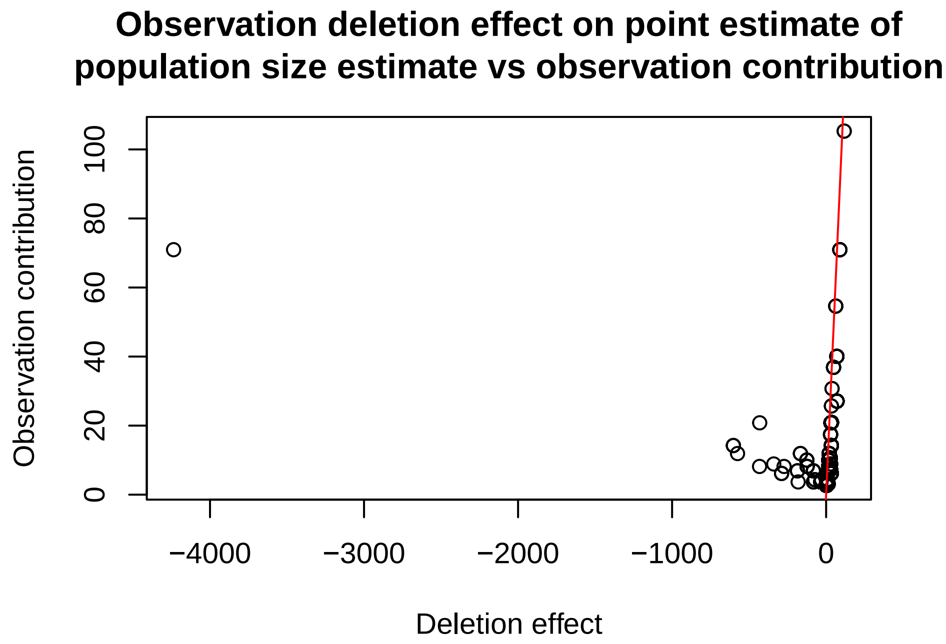
Furthermore, result of the `dfbeta` can be further used in the function `dfpopsize` which allows for quantification of LOO on the population size.

```
R> dfp <- dfpopsize(basicModel, dfbeta = dfb)
R> knitr::kable(as.data.frame(t(matrix(
+   summary(dfp), dimnames = list(attr(summary(dfp), "names"), 1)
+ ))))
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-4236.412	2.663536	2.663536	5.448042	17.28424	117.4479

The comparison of deletion effect on population size estimate and inverse probability weights, which refer to the contribution of a given observation to the population size estimation, is presented in the Figure bellow:

```
R> plot(basicModel, plotType = "dfpopContr", dfpop = dfp)
```



This plot informs on the change of the population size if a given observation will be removed. For instance if we remove observation 542 from the data then population size will rise by about 4236.

The full list of plot types along with the list of optional arguments which may be passed from the call to the `plot` method down to base R and **graphics** functions is listed in the help file

```
R> ?plot.singleRStaticCountData
```

3.6. The stratifyPopsize method

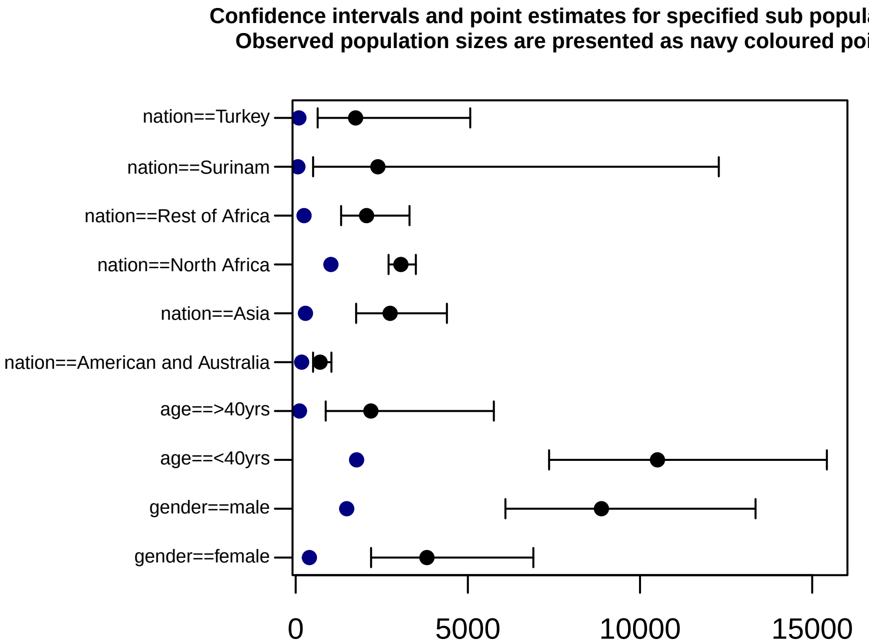
Researchers may be interested on only in the total population size but also in specific sub-populations (e.g. males, females, group pages). For that reason we have created function `stratifyPopsize` which allows to estimate the size by stratas defined by the coefficients in the model (the default option).

```
R> popSizeStratas <- stratifyPopsize(basicModel)
R>
R> cols <- c("name", "Observed", "Estimated", "logNormalLowerBound",
+            "logNormalUpperBound")
R> cols_custom <- c("Name", "Obs", "Estimated", "LowerBound", "UpperBound")
R>
R> knitr::kable(popSizeStratas[, cols],  col.names = cols_custom, digits = 2)
```

Name	Obs	Estimated	LowerBound	UpperBound
gender==female	398	3811.09	2189.04	6902.14
gender==male	1482	8879.26	6090.78	13354.89
age==<40yrs	1769	10506.90	7359.41	15426.47
age==>40yrs	111	2183.45	872.01	5754.88
nation==American and Australia	173	708.37	504.61	1037.33
nation==Asia	284	2742.31	1755.25	4391.59
nation==North Africa	1023	3055.20	2697.49	3489.33
nation==Rest of Africa	243	2058.15	1318.75	3305.79
nation==Surinam	64	2386.45	505.25	12288.01
nation==Turkey	93	1739.86	638.05	5068.96

One may also specify `plotType = "strata"` in the `plot` function which results in a plot with point and CI estimates of the population size.

```
R> par(mar = c(2.5, 8.5, 4.1, 2.5), cex.main = .7, cex.lab = .6)
R> plot(basicModel, plotType = "strata")
```



The method for `singleRStaticCountData` class accepts three optional parameters `stratas`, `alpha`, `cov` which correspond to specification of sub populations, the significance levels and the covariance matrix that will be used to compute standard errors. An example of the full call is presented below.

```
R> library(sandwich)
R> popSizeStratasCustom <- stratifyPopsize(
```

```
+   object  = basicModel,
+   stratas = ~ gender / (nation + age),
+   alpha   = rep(c(.1, .2, .3, .4, .5), length.out = 18),
+   cov     = vcovHC(basicModel, type = "HC4")
+ )
R>
R> knitr::kable(popSizeStratasCustom[, cols], col.names = cols_custom, digits=2)
```

Name	Obs	Estimated	LowerBound	UpperBound
gender==female	398	3811.09	2275.64	6602.17
gender==male	1482	8879.26	6745.57	11877.89
genderfemale:nationAmerican and Australia	67	328.88	255.77	430.30
gendermale:nationAmerican and Australia	106	379.49	318.47	458.03
genderfemale:nationAsia	62	775.91	604.53	1001.42
gendermale:nationAsia	222	1966.41	1225.65	3253.90
genderfemale:nationNorth Africa	169	644.05	517.56	816.45
gendermale:nationNorth Africa	854	2411.15	2242.89	2599.80
genderfemale:nationRest of Africa	65	682.18	527.30	888.94
gendermale:nationRest of Africa	178	1375.98	1155.80	1645.73
genderfemale:nationSurinam	20	931.47	234.36	3895.64
gendermale:nationSurinam	44	1454.99	502.11	4389.90
genderfemale:nationTurkey	15	448.61	241.64	844.56
gendermale:nationTurkey	78	1291.25	836.66	2018.24
genderfemale:age<40yrs	378	3169.83	2617.47	3858.42
gendermale:age<40yrs	1391	7337.07	5543.48	9905.38
genderfemale:age>40yrs	20	641.27	288.73	1456.27
gendermale:age>40yrs	91	1542.19	899.88	2694.54

We provide integration with the **sandwich** (Zeileis, Köll, and Graham 2020) package to correct variance-covariance matrix in the δ method. In the code we have used the `vcovHC` method for `singleRStaticCountData` class from the **sandwich** package, different significance levels for confidence intervals in each strata and a formula to specify that we wanted estimates for both males and females subdivided by `nation` and `age`. The `stratas` parameter may be specified either as:

- a formula with empty left hand side which we have seen here,
- a logical vector with number of entries equal to number of rows in the dataset in which case only one strata will be created,
- a (named) list where each element is a logical vector, names of the list will be used to specify names variable in returned object,
- a vector of names of explanatory variables which will result in every level of explanatory variable having its own sub population for each variable specified,

- or not supplied at all in which case stratas will correspond to levels of each factor in the data without any interactions (string vectors will be converted to factors for the convenience of the user).

For plotting only the `logNormal` type of confidence interval is used since the studentized confidence intervals often result in negative lower bounds.

3.7. Implementation of Variance estimation

The package implements analytic and bootstrap variance estimators. In the control function `controlPopVar` user may specify the `bootType` argument which has three possible values `"parametric", "semi-parametric"` and `"nonparametric"` Additional arguments accepted by the `contorlPopVar` function which are relevant to bootstrap are:

- `alpha, B` – significance level and number of bootstrap samples to be performed respectively with 0.05 and 500 being the default options.
- `cores` – number of process cores to use in bootstrap (1 by default) parallel computing is done via **doParallel, foreach, parallel** packages.
- `keepbootStat` – logical value indicating whether to keep a vector of statistics produced by bootstrap.
- `traceBootstrapSize, bootstrapVisualTrace` – logical values indicating whether sample and population size should be tracked (`FALSE` by default) these work only when `cores` = 1.
- `fittingMethod, bootstrapFitcontrol` – fitting method (by default the same as used in the original call) and control parameters (`controlMethod`) for model fitting in bootstrap.

4. Integration with the VGAM, countreg packages

As noted at the beginning we provide an integration with the **VGAM** and **countreg** packages via the **singleRcaptureExtra** package available through Github at https://github.com/ncn-foreigners/singleRcaptureExtra.

```
R> install.packages("pak")
R> pak::pak("ncn-foreigners/singleRcaptureExtra")
```

The **singleRcaptureExtra** allows for converting objects created by `vglm, vgam, countreg` functions from packages **VGAM, countreg** to a `singleRStaticCountData` via the respective `estimatePopsize` methods for their classes. The help files for all the methods and all the control functions are accessed by

```
R> ?estimatePopsize.vgam
R> ?controlEstPopVgam
```

Using the fitted `zerotrunc, vglm, vgam` class objects in population size estimation such as the one additive models with smooth terms for dataset from Böhning *et al.* (2013).

```
R> library(VGAM)
R> library(singleRcaptureExtra)
R> modelVgam <- vgam(
+   TOTAL_SUB ~ (s(log_size, df  = 3) + s(log_distance, df  = 2)) / C_TYPE,
+   data = farmsubmission,
+   # Using different link since
+   # VGAM uses parametrisation with 1/alpha
+   family = posnegbinomial(
+     lsize = negloglink
+   )
+ )
```

Estimation of the population size can be accomplished with the following syntax simple syntax.

```
R> modelVgamPop <- estimatePopsize(modelVgam)
```

The resulting object is of class `singleRforeign` to underline that the parameters were estimated outside the **singleRcapture**. The structure of the object is as follows

```
R> str(modelVgamPop,1)
```

```
List of 5
 $ foreignObject :Formal class 'vgam' [package "VGAM"] with 43 slots
 $ call          : language estimatePopsize.vgam(formula = modelVgam)
 $ sizeObserved  : int 12036
 $ populationSize:List of 5
  ..- attr(*, "class")= chr "popSizeEstResults"
 $ derivFunc     :function (eta)
 - attr(*, "class")= chr [1:4] "singleRadditive" "singleRforeign" "singleRStaticCountData"
```

Compare with a similar linear model from base **singleRcapture**:

```
R> modelBase <- estimatePopsize(
+   TOTAL_SUB ~ (log_size + log_distance) * C_TYPE,
+   data = farmsubmission,
+   model = ztnegbin()
+ )
R> summary(modelBase)
```

```
Call:
estimatePopsize.default(formula = TOTAL_SUB ~ (log_size + log_distance) *
    C_TYPE, data = farmsubmission, model = ztnegbin())
```

```
Pearson Residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.729357 -0.317558 -0.152482  0.000609  0.148985  6.604269

Coefficients:
-----------------------
For linear predictors associated with: lambda
                         Estimate Std. Error z value  P(>|z|)
(Intercept)              -1.77609    0.45894  -3.870 0.000109 ***
log_size                  0.49391    0.02521  19.594  < 2e-16 ***
log_distance             -0.14106    0.04098  -3.442 0.000578 ***
C_TYPEDairy              -1.68591    0.55327  -3.047 0.002310 **
log_size:C_TYPEDairy      0.26504    0.03495   7.583 3.37e-14 ***
log_distance:C_TYPEDairy  0.08568    0.04874   1.758 0.078762 .
-----------------------
For linear predictors associated with: alpha
            Estimate Std. Error z value  P(>|z|)
(Intercept)  0.57673    0.07267   7.936 2.09e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

AIC: 34481.99
BIC: 34533.76
Residual deviance: 17611.16

Log-likelihood: -17233.99 on 24065 Degrees of freedom
Number of iterations: 9
-----------------------
Population size estimation results:
Point estimate 38877
Observed proportion: 31% (N obs = 12036)
Std. Error 1749.448
95% CI for the population size:
          lowerBound upperBound
normal      35448.14   42305.85
logNormal   35661.32   42530.37
95% CI for the share of observed population:
          lowerBound upperBound
normal      28.44996   33.95382
logNormal   28.29978   33.75085

R> summary(modelVgamPop)


Call:
estimatePopsize.vgam(formula = modelVgam)

-----------------------
Population size estimation results:
Point estimate 37760.01
Observed proportion: 31.9% (N obs = 12036)
Std. Error 1630.429
95% CI for the population size:
```

```
          lowerBound upperBound
normal      34564.42   40955.59
logNormal   34757.77   41158.93
95% CI for the share of observed population:
          lowerBound upperBound
normal      29.38793   34.82193
logNormal   29.24274   34.62823

-------------------------------
-- Summary of foreign object --
-------------------------------

Call:
vgam(formula = TOTAL_SUB ~ (s(log_size, df = 3) + s(log_distance,
    df = 2))/C_TYPE, family = posnegbinomial(lsize = negloglink),
    data = farmsubmission)

Names of additive predictors: loglink(munb), negloglink(size)

Dispersion Parameter for posnegbinomial family:   1

Log-likelihood: -17214.62 on 24063.17 degrees of freedom

Number of Fisher scoring iterations:  11

DF for Terms and Approximate Chi-squares for Nonparametric Effects

                                                   Df Npar Df Npar Chisq
(Intercept):1                                       1
(Intercept):2                                       1
s(log_size, df = 3)                                 1     1.8     51.949
s(log_distance, df = 2)                             1     1.0      3.503
s(log_size, df = 3):s(log_distance, df = 2):C_TYPE  2
                                                     P(Chi)
(Intercept):1
(Intercept):2
s(log_size, df = 3)                                0.000000
s(log_distance, df = 2)                            0.063835
s(log_size, df = 3):s(log_distance, df = 2):C_TYPE
```

5. Concluding remarks

Package **singleRcapture**

... something more on the conclusions

6. Acknowledgements

The authors' work has been financed by the National Science Centre in Poland, OPUS 20, grant no. 2020/39/B/HS4/00941.

The authors would like to thank Layna Dennett from University of Southampton for usefull comments that led to the improved of the functionaly of the package.

A. Detailed information

A.1. The estimatePopsizeFit function

```
R> X <- matrix(data = 0, nrow = 2 * NROW(farmsubmission), ncol = 7)
R> X[1:NROW(farmsubmission), 1:4] <- model.matrix(
+   ~ 1 + log_size + log_distance + C_TYPE,
+   farmsubmission
+ )
R> X[-(1:NROW(farmsubmission)), 5:7] <- X[1:NROW(farmsubmission), c(1, 3, 4)]
R> # this attribute tells the function which elements of the design matrix
R> # correspond to which linear predictor
R> attr(X, "hwm") <- c(4, 3)
R> start <- glm.fit(# get starting points
+   y = farmsubmission$TOTAL_SUB,
+   x = X[1:NROW(farmsubmission), 1:4],
+   family = poisson()
+ )$coefficients
R> res <- estimatePopsizeFit(
+   y            = farmsubmission$TOTAL_SUB,
+   X            = X,
+   method       = "IRLS",
+   priorWeights = 1,
+   family       = ztoigeom(),
+   control      = controlMethod(silent = TRUE),
+   coefStart    = c(start, 0, 0, 0),
+   etaStart     = matrix(X %*% c(start, 0, 0, 0), ncol = 2),
+   offset       = cbind(rep(0, NROW(farmsubmission)),
+                        rep(0, NROW(farmsubmission)))
+ )# extract results
R> ll <- ztoigeom()$makeMinusLogLike(y = farmsubmission$TOTAL_SUB, X = X)
R> print(c(res$beta, -ll(res$beta), res$iter))

[1] -2.784523e+00  6.170270e-01 -6.455925e-02  5.346108e-01 -3.174491e+00
[6]  1.280589e-01 -1.086452e+00 -1.727876e+04  1.500000e+01

R> # Compare with optim call
R> res2 <- estimatePopsizeFit(
+   y = farmsubmission$TOTAL_SUB,
+   X = X,
+   method = "optim",
+   priorWeights = 1,
```

```
+   family = ztoigeom(),
+   coefStart = c(start, 0, 0, 0),
+   control = controlMethod(silent = TRUE),
+   offset = cbind(rep(0, NROW(farmsubmission)), rep(0, NROW(farmsubmission)))
+ )# extract results
R> c(res2$beta, -ll(res2$beta), res2$iter)
```

```
-2.640779e+00  6.258275e-01 -8.293688e-02  5.324707e-01 -1.243731e-01
                                               function      gradient
-1.629884e-01 -1.105502e+00 -1.728034e+04  1.002000e+03            NA
```

A.2. Structure of a family function

- makeMinusLogLike – A factory function for creating the:

$$\ell(\boldsymbol{\beta}), \frac{\partial \ell}{\partial \boldsymbol{\beta}}, \frac{\partial^2 \ell}{\partial \boldsymbol{\beta}^\top \partial \boldsymbol{\beta}}$$

  functions from $\boldsymbol{y}$ vector and $\boldsymbol{X}_{vlm}$ the argument deriv with possible values in c(0, 1, 2) provides which derivative to return with the default 0 being just the minus log-likelihood.

- links – List with link functions.

- mu.eta, variance – Functions of linear predictors that return expected value and variance. There is a ‘type‘ argument with 2 possible values "trunc" and "nontrunc" that specifies whether to return $\mathbb{E}[Y|Y>0], \mathrm{var}[Y|Y>0]$ or $\mathbb{E}[Y], \mathrm{var}[Y]$ respectively, also the deriv argument with values in c(0, 1, 2) is used for indicating the derivative with respect to the linear predictors with is used for providing standard error in predict method.

- family – Character that specifies name of the model.

- valideta, validmu – For now only returns true. In near future will be used to check whether applied linear predictors are valid (i.e. are transformed into some elements of parameter space the subjected to inverse link function).

- funcZ, Wfun – Functions that create pseudo residuals and working weights used in IRLS algorithm.

- devResids – Function that given the linear predictors prior weights vector and response vector returns deviance residuals.

- pointEst, popVar – Functions that given prior weights linear predictors and in the later case also estimation of $\mathrm{cov}(\hat{\boldsymbol{\beta}})$ and $\boldsymbol{X_{vlm}}$ matrix return point estimate for population size and analytic estimation of its variance.There is a additional boolean parameter contr in the former function that if set to true returns contribution of each unit.

- `etaNames` – Names of linear predictors.
- `densityFunction` – A function that given linear predictors returns value of PMF at values `x`. Additional argument `type` specifies whether to return $\mathbb{P}[Y|Y>0]$ or $\mathbb{P}[Y]$.
- `simulate` – A function that generates values of dependent vector given linear predictors.
- `getStart` – Expression for generating starting points.

B. Implementing custom singleRcapture family function

Suppose we want to implement a very specific zero truncated family function in the **singleRcapture** which corresponds to the following "untruncated" distribution:

$$\mathbb{P}[Y=y|\lambda,\pi] = \begin{cases} 1-\frac{1}{2}\lambda-\frac{1}{2}\pi & \text{when: } y=0 \\ \frac{1}{2}\pi & \text{when: } y=1 \\ \frac{1}{2}\lambda & \text{when: } y=2, \end{cases} \tag{4}$$

with $\lambda, \pi \in (0,1)$ being dependent on covariates. The following would be one way of implementing it, with `lambda, pi` in the code meaning $\frac{1}{2}\lambda, \frac{1}{2}\pi$ in the equation above:

```
R> myFamilyFunction <- function(lambdaLink = c("logit", "cloglog", "probit"),
+                               piLink     = c("logit", "cloglog", "probit"),
+                               ...) {
+   if (missing(lambdaLink)) lambdaLink <- "logit"
+   if (missing(piLink))         piLink <- "logit"
+
+   links <- list()
+   attr(links, "linkNames") <- c(lambdaLink, piLink)
+
+   lambdaLink <- switch(lambdaLink,
+     "logit"   = singleRcapture:::singleRinternallogitLink,
+     "cloglog" = singleRcapture:::singleRinternalcloglogLink,
+     "probit"  = singleRcapture:::singleRinternalprobitLink
+   )
+
+   piLink <- switch(piLink,
+     "logit"   = singleRcapture:::singleRinternallogitLink,
+     "cloglog" = singleRcapture:::singleRinternalcloglogLink,
+     "probit"  = singleRcapture:::singleRinternalprobitLink
+   )
+
+   links[1:2] <- c(lambdaLink, piLink)
+
+   mu.eta <- function(eta, type = "trunc", deriv = FALSE, ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     if (!deriv) {
+       switch (type,
+         "nontrunc" = pi + 2 * lambda,
+         "trunc" = 1 + lambda / (pi + lambda)
+       )
+     } else {
```

```
+       # Only necessary if one wishes to use standard errors in predict method
+       switch (type,
+         "nontrunc" = {
+           matrix(c(2, 1) * c(
+             lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2,
+                 piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2
+           ), ncol = 2)
+         },
+         "trunc" = {
+           matrix(c(
+             pi / (pi + lambda) ^ 2,
+             -lambda / (pi + lambda) ^ 2
+           ) * c(
+             lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2,
+                 piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2
+           ), ncol = 2)
+         }
+       )
+     }
+   }
+
+   variance <- function(eta, type = "nontrunc", ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     switch (type,
+     "nontrunc" = pi * (1 - pi) + 4 * lambda * (1 - lambda - pi),
+     "trunc" = lambda * (1 - lambda) / (pi + lambda)
+     )
+   }
+
+   Wfun <- function(prior, y, eta, ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     G01 <- ((lambda + pi) ^ (-2)) * piLink(eta[, 2], inverse = TRUE, deriv = 1) *
+       lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) * prior / 4
+
+     G00 <- ((lambda + pi) ^ (-2)) - (pi ^ (-2)) - lambda / ((lambda + pi) * (pi ^ 2))
+     G00 <- G00 * prior * (piLink(eta[, 2], inverse = TRUE, deriv = 1) ^ 2) / 4
+
+     G11 <- ((lambda + pi) ^ (-2)) - (((lambda + pi) * lambda) ^ -1)
+     G11 <- G11 * prior * (lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) ^ 2) / 4
+
+     matrix(
+       -c(G11, # lambda
+          G01, # mixed
+          G01, # mixed
+          G00  # pi
+       ),
+       dimnames = list(rownames(eta), c("lambda", "mixed", "mixed", "pi")),
+       ncol = 4
+     )
+   }
+
+   funcZ <- function(eta, weight, y, prior, ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
```

```
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     weight <- weight / prior
+
+     G0 <- (2 - y) / pi     - ((lambda + pi) ^ -1)
+     G1 <- (y - 1) / lambda - ((lambda + pi) ^ -1)
+
+     G1 <- G1 * lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2
+     G0 <- G0 *     piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2
+
+     uMatrix <- matrix(c(G1, G0), ncol = 2)
+
+     weight <- lapply(X = 1:nrow(weight), FUN = function (x) {
+       matrix(as.numeric(weight[x, ]), ncol = 2)
+     })
+
+     pseudoResid <- sapply(X = 1:length(weight), FUN = function (x) {
+       #xx <- chol2inv(chol(weight[[x]])) # less computationally demanding
+       xx <- solve(weight[[x]]) # more stable
+       xx %*% uMatrix[x, ]
+     })
+     pseudoResid <- t(pseudoResid)
+     dimnames(pseudoResid) <- dimnames(eta)
+     pseudoResid
+   }
+
+   minusLogLike <- function(y, X, offset,
+                            weight    = 1,
+                            NbyK      = FALSE,
+                            vectorDer = FALSE,
+                            deriv     = 0,
+                            ...) {
+     y <- as.numeric(y)
+     if (is.null(weight)) {
+       weight <- 1
+     }
+     if (missing(offset)) {
+       offset <- cbind(rep(0, NROW(X) / 2), rep(0, NROW(X) / 2))
+     }
+
+     if (!(deriv %in% c(0, 1, 2))) stop("Only score function and derivatives up to 2 are supported.")
+     deriv <- deriv + 1 # to make it conform to how switch in R works, i.e. indexing begins with 1
+
+     switch (deriv,
+       function(beta) {
+         eta <- matrix(as.matrix(X) %*% beta, ncol = 2) + offset
+         pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+         lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+         -sum(weight * ((2 - y) * log(pi) + (y - 1) * log(lambda) - log(pi + lambda)))
+       },
+       function(beta) {
+         eta <- matrix(as.matrix(X) %*% beta, ncol = 2) + offset
+         pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+         lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+         G0 <- (2 - y) / pi     - ((lambda + pi) ^ -1)
+         G1 <- (y - 1) / lambda - ((lambda + pi) ^ -1)
```

```
+
+         G1 <- G1 * weight * lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2
+         G0 <- G0 * weight *     piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2
+
+         if (NbyK) {
+           XX <- 1:(attr(X, "hwm")[1])
+           return(cbind(as.data.frame(X[1:nrow(eta), XX]) * G1, as.data.frame(X[-(1:nrow(eta)), -XX]) *
+         }
+         if (vectorDer) {
+           return(cbind(G1, G0))
+         }
+
+         as.numeric(c(G1, G0) %*% X)
+       },
+       function (beta) {
+         lambdaPredNumber <- attr(X, "hwm")[1]
+         eta <- matrix(as.matrix(X) %*% beta, ncol = 2) + offset
+         pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+         lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+         res <- matrix(nrow = length(beta), ncol = length(beta),
+                       dimnames = list(names(beta), names(beta)))
+
+         # pi^2 derivative
+         dpi <- (2 - y) / pi - (lambda + pi) ^ -1
+         G00 <- ((lambda + pi) ^ (-2)) - (2 - y) / (pi ^ 2)
+
+         G00 <- t(as.data.frame(X[-(1:(nrow(X) / 2)), -(1:lambdaPredNumber)] *
+         (G00 * ((piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2) ^ 2) +
+         dpi * piLink(eta[, 2], inverse = TRUE, deriv = 2) / 2) * weight)) %*%
+         as.matrix(X[-(1:(nrow(X) / 2)), -(1:lambdaPredNumber)])
+         # mixed derivative
+         G01 <- (lambda + pi) ^ (-2)
+
+         G01 <- t(as.data.frame(X[1:(nrow(X) / 2), 1:lambdaPredNumber]) *
+         G01 * (lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2) *
+         (piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2) * weight) %*%
+         as.matrix(X[-(1:(nrow(X) / 2)), -(1:lambdaPredNumber)])
+         # lambda^2 derivative
+         G11 <- ((lambda + pi) ^ (-2)) - (y - 1) / (lambda ^ 2)
+         dlambda <- (y - 1) / lambda - ((lambda + pi) ^ -1)
+
+         G11 <- t(as.data.frame(X[1:(nrow(X) / 2), 1:lambdaPredNumber] *
+         (G11 * ((lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2) ^ 2) +
+         dlambda * lambdaLink(eta[, 1], inverse = TRUE, deriv = 2) / 2) * weight)) %*%
+         X[1:(nrow(X) / 2), 1:lambdaPredNumber]
+
+         res[-(1:lambdaPredNumber), -(1:lambdaPredNumber)] <- G00
+         res[1:lambdaPredNumber, 1:lambdaPredNumber] <- G11
+         res[1:lambdaPredNumber, -(1:lambdaPredNumber)] <- t(G01)
+         res[-(1:lambdaPredNumber), 1:lambdaPredNumber] <- G01
+
+         res
+       }
+     )
+   }
+
```

```
+   validmu <- function(mu) {
+     (sum(!is.finite(mu)) == 0) && all(0 < mu) && all(2 > mu)
+   }
+
+   # this is optional
+   devResids <- function(y, eta, wt, ...) {
+     0
+   }
+
+   pointEst <- function (pw, eta, contr = FALSE, ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+     N <- pw / (lambda + pi)
+     if(!contr) {
+       N <- sum(N)
+     }
+     N
+   }
+
+   popVar <- function (pw, eta, cov, Xvlm, ...) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     bigTheta1 <- -pw / (pi + lambda) ^ 2 # w.r to pi
+     bigTheta1 <- bigTheta1 * piLink(eta[, 2], inverse = TRUE, deriv = 1) / 2
+     bigTheta2 <- -pw / (pi + lambda) ^ 2 # w.r to lambda
+     bigTheta2 <- bigTheta2 * lambdaLink(eta[, 1], inverse = TRUE, deriv = 1) / 2# w.r to lambda
+
+     bigTheta <- t(c(bigTheta2, bigTheta1) %*% Xvlm)
+
+     f1 <- t(bigTheta) %*% as.matrix(cov) %*% bigTheta
+
+     f2 <- sum(pw * (1 - pi - lambda) / ((pi + lambda) ^ 2))
+
+     f1 + f2
+   }
+
+   dFun <- function (x, eta, type = c("trunc", "nontrunc")) {
+     if (missing(type)) type <- "trunc"
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+
+     switch (type,
+       "trunc" = {
+         (pi * as.numeric(x == 1) + lambda * as.numeric(x == 2)) / (pi + lambda)
+       },
+       "nontrunc" = {
+         (1 - pi - lambda) * as.numeric(x == 0) +
+         pi * as.numeric(x == 1) + lambda * as.numeric(x == 2)
+       }
+     )
+   }
+
+   simulate <- function(n, eta, lower = 0, upper = Inf) {
+     pi     <-     piLink(eta[, 2], inverse = TRUE) / 2
+     lambda <- lambdaLink(eta[, 1], inverse = TRUE) / 2
+     CDF <- function(x) {
```

```
+       ifelse(x == Inf, 1,
+       ifelse(x < 0, 0,
+       ifelse(x < 1, 1 - pi - lambda,
+       ifelse(x < 2, 1 - lambda, 1))))
+     }
+     lb <- CDF(lower)
+     ub <- CDF(upper)
+     p_u <- stats::runif(n, lb, ub)
+     sims <- rep(0, n)
+     cond <- CDF(sims) <= p_u
+     while (any(cond)) {
+       sims[cond] <- sims[cond] + 1
+       cond <- CDF(sims) <= p_u
+     }
+     sims
+   }
+
+   getStart <- expression(
+     if (method == "IRLS") {
+       etaStart <- cbind(
+         family$links[[1]](mean(observed == 2) * (1 + 0 * (observed == 2))), # lambda
+         family$links[[2]](mean(observed == 1) * (1 + 0 * (observed == 1)))  # pi
+       ) + offset
+     } else if (method == "optim") {
+       init <- c(
+         family$links[[1]](weighted.mean(observed == 2, priorWeights) * 1 + .0001),
+         family$links[[2]](weighted.mean(observed == 1, priorWeights) * 1 + .0001)
+       )
+       if (attr(terms, "intercept")) {
+         coefStart <- c(init[1], rep(0, attr(Xvlm, "hwm")[1] - 1))
+       } else {
+         coefStart <- rep(init[1] / attr(Xvlm, "hwm")[1], attr(Xvlm, "hwm")[1])
+       }
+       if ("(Intercept):pi" %in% colnames(Xvlm)) {
+         coefStart <- c(coefStart, init[2], rep(0, attr(Xvlm, "hwm")[2] - 1))
+       } else {
+         coefStart <- c(coefStart, rep(init[2] / attr(Xvlm, "hwm")[2], attr(Xvlm, "hwm")[2]))
+       }
+     }
+   )
+
+   structure(
+     list(
+       makeMinusLogLike = minusLogLike,
+       densityFunction  = dFun,
+       links     = links,
+       mu.eta    = mu.eta,
+       valideta  = function (eta) {TRUE},
+       variance  = variance,
+       Wfun      = Wfun,
+       funcZ     = funcZ,
+       devResids = devResids,
+       validmu   = validmu,
+       pointEst  = pointEst,
+       popVar    = popVar,
+       family    = "myFamilyFunction",
+       etaNames  = c("lambda", "pi"),
```

```
+       simulate  = simulate,
+       getStart  = getStart,
+       extraInfo = c(
+         mean       = "pi / 2 + lambda",
+         variance   = paste0("(pi / 2) * (1 - pi / 2) + 2 * lambda * (1 - lambda / 2 - pi / 2)"),
+         popSizeEst = "(1 - (pi + lambda) / 2) ^ -1",
+         meanTr     = "1 + lambda / (pi + lambda)",
+         varianceTr = paste0("lambda * (1 - lambda / 2) / (pi + lambda)")
+       )
+     ),
+     class = c("singleRfamily", "family")
+   )
+ }
```

A quick tests shows us that this implementation in fact works:

```
R> set.seed(123)
R> Y <- simulate(
+     myFamilyFunction(lambdaLink = "logit", piLink = "logit"),
+     nsim = 1000, eta = matrix(0, nrow = 1000, ncol = 2),
+     truncated = FALSE
+ )
R> mm <- estimatePopsize(
+     formula = Y ~ 1,
+     data = data.frame(Y = Y[Y > 0]),
+     model = myFamilyFunction(lambdaLink = "logit",
+                              piLink = "logit"),
+     # the usual observed information matrix
+     # is ill-suited for this distribution
+     controlPopVar = controlPopVar(covType = "Fisher")
+ )
R> summary(mm)


Call:
estimatePopsize.default(formula = Y ~ 1, data = data.frame(Y = Y[Y >
    0]), model = myFamilyFunction(lambdaLink = "logit", piLink = "logit"),
    controlPopVar = controlPopVar(covType = "Fisher"))

Pearson Residuals:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.8198 -0.8198  0.8099  0.0000  0.8099  0.8099

Coefficients:
-----------------------
For linear predictors associated with: lambda
            Estimate Std. Error z value P(>|z|)
(Intercept)  0.01217    0.20253    0.06   0.952
-----------------------
For linear predictors associated with: pi
```

```
            Estimate Std. Error z value P(>|z|)
(Intercept) -0.01217    0.08926  -0.136   0.892

AIC: 687.4249
BIC: 695.8259
Residual deviance: 0

Log-likelihood: -341.7124 on 984 Degrees of freedom
Number of iterations: 2
-----------------------
Population size estimation results:
Point estimate 986
Observed proportion: 50% (N obs = 493)
Std. Error 70.30092
95% CI for the population size:
          lowerBound upperBound
normal      848.2127   1123.787
logNormal   866.3167   1144.053
95% CI for the share of observed population:
          lowerBound upperBound
normal      43.86951   58.12221
logNormal   43.09241   56.90759
```

Where the link functions such as `singleRcapture:::singleRinternalcloglogLink` are just internal functions in **singleRcapture** that compute link functions their inverses and derivatives of both links and inverse link up to third order:

```
R> singleRcapture:::singleRinternalcloglogLink

function (x, inverse = FALSE, deriv = 0)
{
    deriv <- deriv + 1
    if (isFALSE(inverse)) {
        res <- switch(deriv, log(-log(1 - x)), -1/((1 - x) *
            log(1 - x)), -(1 + log(1 - x))/((x - 1)^2 * log(1 -
            x)^2), (2 * log(1 - x)^2 + 3 * log(1 - x) + 2)/(log(1 -
            x)^3 * (x - 1)^3))
    }
    else {
        res <- switch(deriv, 1 - exp(-exp(x)), exp(x - exp(x)),
            (1 - exp(x)) * exp(x - exp(x)), (exp(2 * x) - 3 *
                exp(x) + 1) * exp(x - exp(x)))
    }
    res
}
<bytecode: 0x10f646340>
<environment: namespace:singleRcapture>
```

one might of course include code for computing them manually.

References

Baffour-Awuah B (2009). *Estimation of population totals from imperfect census, survey and administrative records*. Ph.D. thesis.

Baillargeon S, Rivest LP (2007). "Rcapture: loglinear models for capture-recapture in R." *Journal of statistical software*, **19**, 1–31.

Böhning D (2023). "On the equivalence of one-inflated zero-truncated and zero-truncated one-inflated count data likelihoods." *Biometrical Journal*, **65**(2), 2100343.

Böhning D, Bunge J, Heijden PG (2018). *Capture-recapture methods for the social and medical sciences*. CRC Press Boca Raton.

Böhning D, Friedl H (2024). "One-Inflation and Zero-Truncation Count Data Modelling Revisited With a View on Horvitz–Thompson Estimation of Population Size." *International Statistical Review*.

Böhning D, van der Heijden PGM (2009). "A covariate adjustment for zero-truncated approaches to estimating the size of hidden and elusive populations." *The Annals of Applied Statistics*, **3**(2), 595 – 610. doi:10.1214/08-AOAS214.

Böhning D, van der Heijden PGM (2019). "The identity of the zero-truncated, one-inflated likelihood and the zero-one-truncated likelihood for general count densities with an application to drink-driving in Britain." *The Annals of Applied Statistics*, **13**(2), 1198 – 1211. doi:10.1214/18-AOAS1232.

Böhning D, Vidal-Diez A, Lerdsuwansri R, Viwatwongkasem C, Arnold M (2013). "A Generalization of Chao's Estimator for Covariate Information." *Biometrics*, **69**(4), 1033–1042.

Chao A (1987). "Estimating the population size for capture-recapture data with unequal catchability." *Biometrics*, pp. 783–791.

Cruyff MJLF, van der Heijden PGM (2008). "Point and Interval Estimation of the Population Size Using a Zero-Truncated Negative Binomial Regression Model." *Biometrical Journal*, **50**(6), 1035–1050.

Dunne J, Zhang LC (2024). "A system of population estimates compiled from administrative data only." *Journal of the Royal Statistical Society Series A: Statistics in Society*, **187**(1), 3–21.

Godwin RT, Böhning D (2017a). "Estimation of the population size by using the one-inflated positive Poisson model." *Journal of the Royal Statistical Society Series C: Applied Statistics*, **66**(2), 425–448.

Godwin RT, Böhning D (2017b). "Estimation of the population size by using the one-inflated positive Poisson model." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **66**(2), 425–448.

Hayes A, Moller-Trane R, Jordan D, Northrop P, Lang MN, Zeileis A (2024). *distributions3: Probability Distributions as S3 Objects*. R package version 0.2.2, URL `https://CRAN.R-project.org/package=distributions3`.

Kleiber C, Zeileis A (2016). "Visualizing Count Data Regressions Using Rootograms." *The American Statistician*, **70**(3), 296–303. `doi:10.1080/00031305.2016.1173590`.

Norris JL, Pollock KH (1996). "Including model uncertainty in estimating variances in multiple capture studies." *Environmental and Ecological Statistics*, **3**(3), 235–244.

van der Heijden PG, Bustami R, Cruyff MJ, Engbersen G, van Houwelingen HC (2003). "Point and interval estimation of the population size using the truncated Poisson regression model." *Statistical Modelling*, **3**(4), 305–322.

Vincent K, Thompson S (2022). "Estimating the size and distribution of networked populations with snowball sampling." *Journal of Survey Statistics and Methodology*, **10**(2), 397–418.

Wolter KM (1986). "Some coverage error models for census data." *Journal of the American Statistical Association*, **81**(394), 337–346.

Yee TW (2015). *Vector Generalized Linear and Additive Models: With an Implementation in R*. 1st edition. Springer Publishing Company, Incorporated.

Zeileis A, Kleiber C, Jackman S (2008). "Regression Models for Count Data in R." *Journal of Statistical Software*, **27**(8), 1–25. `doi:10.18637/jss.v027.i08`.

Zeileis A, Köll S, Graham N (2020). "Various Versatile Variances: An Object-Oriented Implementation of Clustered Covariances in R." *Journal of Statistical Software*, **95**(1), 1–36. `doi:10.18637/jss.v095.i01`.

Zelterman D (1988). "Robust estimation in truncated discrete distributions with application to capture-recapture experiments." *Journal of statistical planning and inference*, **18**(2), 225–237.

Zhang LC (2019). "A note on dual system population size estimator." *Journal of Official Statistics*, **35**(1), 279–283.

Zwane E, Van der Heijden P (2003). "Implementing the parametric bootstrap in capture–recapture models with continuous covariates." *Statistics & probability letters*, **65**(2), 121–125.

Affiliation:

Piotr Chlebicki
Stockholm University
Matematiska institutionen
Albano hus 1
106 91 Stockholm, Sweden
E-mail: piotr.chlebicki@math.su.se
URL: https://github.com/Kertoo, https://www.su.se/profiles/pich3772

Maciej Beręsewicz
Poznań University of Economics and Business
Statistical Office in Poznań
Poznań University of Economics and Business
Department of Statistics
Institute of Informatics and Quantitative Economics
Al. Niepodległosci 10
61-875 Poznań, Poland

Statistical Office in Poznań
ul. Wojska Polskiego 27/29
60-624 Poznań, Poland
E-mail: maciej.beresewicz@ue.poznan.pl

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd