CensoPy

Statistical computations for exponentially distributed lifetimes under hybrid censoring

Jakob I. Cramer, Erhard Cramer

September 2026
Version 0.1

Contents

1 Documentation **1**
1.1 Classes | Generate (progressive) hybrid censored data . . . 2
1.1.1 Class `CensoPy.C_I_HCS(...)` . . . 2
1.1.2 Class `CensoPy.C_II_HCS(...)` . . . 2
1.1.3 Class `CensoPy.C_I_PCS(...)` . . . 3
1.1.4 Class `CensoPy.C_II_PCS(...)` . . . 3
1.2 Table of hybrid censoring schemes and choice of parameters . . . 3
1.3 Methods | Statistical computations for (progressive) hybrid censored data . . . 3
1.3.1 Method `*.pdf_TTT(...)` . . . 3
1.3.2 Method `*.cdf_TTT(...)` . . . 4
1.3.3 Method `*.pdf_MLE(...)` . . . 4
1.3.4 Method `*.cdf_MLE(...)` . . . 5
1.3.5 Method `*.calc_MLE(...)` . . . 5
1.3.6 Method `*.monte_carlo(...)` . . . 5
1.3.7 Method `CensoPy.ker(...)` . . . 6

2 Applications and Examples **6**

3 Credits **6**

References **7**

The Python package `CensoPy` can be used to compute density functions, cumulative distribution functions, and exact confidence intervals obtained under (progressive) hybrid censoring. The theoretical backgrounds are described in Cramer and Cramer (2026). A recent review on hybrid censoring schemes is provided by Balakrishnan et al. (2023) (see also Cramer (2025)).

1 Documentation

As mentioned in the abstract, the Python package `CensoPy` provides a code support statistical analysis of (progressive) hybrid censored data based on exponentially distributed lifetimes. It provides tools to generate hybrid censored data for any general hybrid censored scheme. Based on this data, exact (conditional) density functions and cumulative distribution functions can be computed and plotted taking into account numerical accuracy (examples are provided on GitLab). Additionally, the computation of confidence intervals based on the 'pivoting the cumulative distribution function method' is implemented. Finally, it can be used to simulate hybrid censored data for exponentially distributed random variables. Of course, using a transformation of the data or a different random number generator, the package can be easily used generate hybrid censored samples for such data.

We provide some example code to illustrate the use of the package. In particular, we use the package matplotlib to generate various plots of the above mentioned quantities (see Section 2).

1.1 Classes | Generate (progressive) hybrid censored data

In hybrid censoring, censoring schemes are categorized in two categories called **C-I**- and **C-II**-hybrid censoring schemes (see Cramer and Cramer (2026)). The package provides four classes depending on the underlying data. We provide **C-I**- and **C-II**-hybrid censoring schemes for order statistics (see Balakrishnan et al. (2023) details) and progressively Type-II censored order statistics (see Balakrishnan and Cramer (2014) for details). These classes generate for a given data the corresponding hybrid censored data.

1.1.1 Class `CensoPy.C_I_HCS(...)`

This class can be used to initialize objects of a **C-I-hybrid censoring scheme**. Use

```
HCS_ex=CensoPy.C_I_HCS(n, thresholds, censoring_numbers, Digits, Method)
```

to initialize the hybrid censoring scheme `HCS_ex`. The parameters are given by:

- `n` (integer): sample size $n > 0$
- `thresholds` (list of k strictly increasing floats): List of thresholds (in ascending order $0 < T_1 < \cdots < T_k$).
- `censoring_numbers` (list of k strictly decreasing integers): List of censoring numbers (in descending order $n \geq m_1 > \cdots > m_k > 0$)
- `Digits` (integer, default-value 200): Sets the number of decimal places for the calculations (`Digits` > 0). If this value is too large, it leads to a loss of speed. If this number is too small, it results in numerical instabilities.
- `Method` (string, default-value `'analytic'`): Method to compute the B-spline basis functions: analytic (`'analytic'`) or recursive (`'recursive'`) using Cox-de Boor recurrence relation.

Please note that the list of both thresholds and censoring numbers must have the same length.

1.1.2 Class `CensoPy.C_II_HCS(...)`

This class can be used to initialize objects of a **C-II-hybrid censoring scheme**. Use

```
HCS_ex=CensoPy.C_II_HCS(n, thresholds, censoring_numbers, Digits, Method)
```

to initialize the hybrid censoring scheme `HCS_ex`. The parameters are given by:

- `n` (integer): sample size $n > 0$
- `thresholds` (list of k strictly increasing floats): List of thresholds (in ascending order $0 < T_1 < \cdots < T_k$). If you want a threshold to be infinite, simply enter `'inf'` in the list.
- `censoring_numbers` (list of k strictly decreasing integers): List of censoring numbers (in descending order $n \geq m_1 > \cdots > m_k > 0$)
- `Digits` (integer, default-value 200): Sets the number of decimal places for the calculations (`Digits` > 0). If this value is too large, it leads to a loss of speed. If this number is too small, it results in numerical instabilities.
- `Method` (string, default-value `'analytic'`): Method to compute the B-spline basis functions: analytic (`'analytic'`) or recursive (`'recursive'`) using Cox-de Boor recurrence relation.

Please note that the list of both thresholds and censoring numbers must have the same length.

1.1.3 Class `CensoPy.C_I_PCS(...)`

This class can be used to initialize objects of a **C-I-progressive hybrid censoring scheme**. Use

```
PCS_ex=CensoPy.C_I_PCS(R, thresholds, censoring_numbers, Digits, Method)
```

to initialize the progressive hybrid censoring scheme `PCS_ex`. The parameters are given by:

- `R` (list of integers): progressive censoring plan with length n (observed sample size) and initial sample size $\gamma_1 = \sum_{i=1}^{n}(R_i + 1)$.
- `thresholds` (list of k strictly increasing floats): List of thresholds (in ascending order $0 < T_1 < \cdots < T_k$).
- `censoring_numbers` (list of k strictly decreasing integers): List of censoring numbers (in descending order $n \geq m_1 > \cdots > m_k > 0$)
- `Digits` (integer, default-value 200): Sets the number of decimal places for the calculations (`Digits` > 0). If this value is too large, it leads to a loss of speed. If this number is too small, it results in numerical instabilities.
- `Method` (string, default-value `'recursive'`): Method to compute the B-spline basis functions: recursive (`'recursive'`) using Cox-de Boor recurrence relation.

Please note that the list of both thresholds and censoring numbers must have the same length.

1.1.4 Class `CensoPy.C_II_PCS(...)`

This class can be used to initialize objects of a **C-II-progressive hybrid censoring scheme**. Use

```
PCS_ex=CensoPy.C_I_PCS(R, thresholds, censoring_numbers, Digits, Method)
```

to initialize the progressive hybrid censoring scheme `PCS_ex`. The parameters are given by:

- `R` (list of integers): progressive censoring plan with length n (observed sample size) and initial sample size $\gamma_1 = \sum_{i=1}^{n}(R_i + 1)$.
- `thresholds` (list of k strictly increasing floats): List of thresholds (in ascending order $0 < T_1 < \cdots < T_k$). If you want a threshold to be infinite, simply enter `'inf'` in the list.
- `censoring_numbers` (list of k strictly decreasing integers): List of censoring numbers (in descending order $n \geq m_1 > \cdots > m_k > 0$)
- `Digits` (integer, default-value 200): Sets the number of decimal places for the calculations (`Digits` > 0). If this value is too large, it leads to a loss of speed. If this number is too small, it results in numerical instabilities.
- `Method` (string, default-value `'recursive'`): Method to compute the B-spline basis functions: recursive (`'recursive'`) using Cox-de Boor recurrence relation.

Please note that the list of both thresholds and censoring numbers must have the same length.

1.2 Table of hybrid censoring schemes and choice of parameters

Table 1 is taken from Cramer and Cramer (2026). It illustrates the form of arrays used for the particular hybrid censoring schemes. For the sake of consistency, both arrays have the same length. For **C-II**-hybrid censoring schemes, the threshold array must be supplemented with the entry `inf` as its final component.

1.3 Methods | Statistical computations for (progressive) hybrid censored data

The four classes described in Sections 1.1.1, 1.1.2, 1.1.3 and 1.1.4 have the following six methods. The * symbol always acts as a placeholder for an object of the relevant class.

1.3.1 Method `*.pdf_TTT(...)`

The method computes the density function of the total time on test statistic based on a (progressive) hybrid censored sample at $t > 0$ for an exponential distributed baseline sample with mean $\vartheta > 0$.

Table 1: Implementation of hybrid censoring schemes and input used in the command.

	Hybrid censoring scheme	k	sample size n	censoring numbers $[m_1, \ldots, m_k]$	thresholds $[T_1, \ldots, T_k]$
C-II-HCS	complete sample	1	`n`	`[n]`	`['inf']`
	Type-II	1	`n`	`[m1]`	`['inf']`
	Type-II hybrid	2	`n`	`[n,m2]`	`[T1,'inf']`
	gen. Type-I hybrid	2	`n`	`[m1,m2]`	`[T1,'inf']`
	uni. Type-II hybrid	3	`n`	`[n,m2,m3]`	`[T1,T2,'inf']`
	uni. Type-IV hybrid	3	`n`	`[m1,m2,m3]`	`[T1,T2,'inf']`
	gen. uni. Type-II hybrid	4	`n`	`[n,m2,m3,m4]`	`[T1,T2,T3,'inf']`
	gen. uni. Type-IV hybrid	4	`n`	`[m1,m2,m3,m4]`	`[T1,T2,T3,'inf']`
C-I-HCS	Type-I	1	`n`	`[n]`	`[T1]`
	Type-I hybrid	1	`n`	`[m1]`	`[T1]`
	gen. Type-II hybrid	2	`n`	`[n,m2]`	`[T1,T2]`
	uni. Type-I hybrid	2	`n`	`[m1,m2]`	`[T1,T2]`
	uni. Type-III hybrid	3	`n`	`[n,m2,m3]`	`[T1,T2,T3]`
	gen. uni. Type-I hybrid	3	`n`	`[m1,m2,m3]`	`[T1,T2,T3]`

Notes: For **C-II**-hybrid censoring schemes, the infinite bound $T_k =$ `inf` has to be written as `'inf'` in the parametrization. The dimension k of the arrays is automatically determined from the input.

Usage `*.pdf_TTT(t, theta)`

Input

- `t` (float) argument $t > 0$ of density function
- `theta` (float) True distribution parameter ($\vartheta > 0$)

The `return`-value is a float.

1.3.2 Method `*.cdf_TTT(...)`

The method computes the cumulative distribution function of the total time on test statistic based on a (progressive) hybrid censored sample at $t > 0$ for an exponential distributed baseline sample with mean $\vartheta > 0$.

Usage `*.cdf_TTT(t, theta)`

Input

- `t` (float) argument $t > 0$ of density function
- `theta` (float) True distribution parameter ($\vartheta > 0$)

The `return`-value is a float.

1.3.3 Method `*.pdf_MLE(...)`

The method computes the density function of the maximum likelihood estimator (MLE) based on a (progressive) hybrid censored sample at $t > 0$ for an exponential distributed baseline sample with mean $\vartheta > 0$.

Usage `*.pdf_MLE(t, theta)`

Input

- `t` (float) argument $t > 0$ of density function
- `theta` (float) True distribution parameter ($\vartheta > 0$)

The `return`-value is a float.

1.3.4 Method `*.cdf_MLE(...)`

The method computes the cumulative distribution function of the the maximum likelihood estimator (MLE) based on a (progressive) hybrid censored sample at $t > 0$ for an exponential distributed baseline sample with mean $\vartheta > 0$.

Usage `*.cdf_MLE(t, theta)`

Input

- `t` (float) argument $t > 0$ of density function
- `theta` (float) True distribution parameter ($\vartheta > 0$)

The `return`-value is a float.

1.3.5 Method `*.calc_MLE(...)`

The method computes the the maximum likelihood estimator (MLE) based on a (progressive) hybrid censored sample at $t > 0$ for an exponential distributed baseline sample with mean $\vartheta > 0$.

Usage `*.calc_MLE(data)`

Input

- `data` (list of integers)

Output `(D_HCS, W_HCS, TTT_D_HCS, theta_hat)`

- `D_HCS` (float) sample size of (progressive hybrid censored sample)
- `W_HCS` (float) stopping time
- `TTT_D_HCS` (float) total time on test
- `theta_hat` (float) maximum likelihood estimator for ϑ (if exist, that is, `D_HCS`> 0; otherwise 0 is returned)

1.3.6 Method `*.monte_carlo(...)`

Simulates N independent samples of independent and identically distributed random variables from an exponential distribution with mean $\vartheta > 0$ of size n and applies the (progressive) hybrid censoring to the sample. In case of a progressive hybrid censoring scheme, a progressively Type-II censored sample with sample size n and censoring plan $(R_1, \ldots, R_n)$ is generated using the algorithm of Balakrishnan and Sandhu (1995) (see also Balakrishnan and Cramer (2014, Chapter 8)).

Usage `*.monte_carlo(theta, N)`

Input

- `theta` (float) True distribution parameter ($\vartheta > 0$)
- `N` (integer) Number of samples ($N > 0$)

Output `(D_HCS_array, W_HCS_array, TTT_D_HCS_array, theta_array, q, data)`

- `D_HCS_array` (list of floats) sample size of (progressive hybrid censored sample)
- `W_HCS_array` (list of floats) stopping time
- `TTT_D_HCS_array` (list of floats) total time on test
- `theta_array` (list of floats) maximum likelihood estimator for ϑ (if exist, that is, `D_HCS`> 0; otherwise 0 is returned)
- `q` (integer) count of empty samples
- `data` (2d list of integers) simulated hybrid censored data

1.3.7 Method `CensoPy.ker(...)`

The method provides a histogram and kernel estimate for a hybrid censored data.

Usage `CensoPy.ker(X, x, kernel, bandwidth)`

Input

- `X` (list of floats) data
- `x` (float) argument of kernel function
- `kernel` (string, default 'gaussian') kernel used in the estimation process. For further options of kernel choices, we refer to the documentation of the *sklearn* package.
- `bandwidth` (float, default 0.03) bandwith of kernel function

The `return`-value is a float.
This method is based on the *sklearn* package; more specifically, on the *KernelDensity* method.

2 Applications and Examples

The source code and some example applications for calculating confidence intervals, creating plots, etc. can be downloaded from GitLab. It is also possible to use this package in the R programming language by utilising the R package 'retinculate'. However, as the package is written in Python, a compatible version of Python must be installed on the computer. A brief example of how to create plots in R using our package can also be found on is available at GitLab https://censopy-738e78.pages.git.nrw/.

3 Credits

The package has been coded in Python 3.13. The following packages have been used in the implementation of the package `CensoPy`:

1. numpy: Harris et al. (2020)
   `CensoPy` requires numpy $\geq$ 2.3.5
2. mpmath: The mpmath development team (2026)
   `CensoPy` requires mpmath $\geq$ 1.3.0
3. sklearn: Pedregosa et al. (2011)
   `CensoPy` requires scikit-learn $\geq$ 1.7.2

References

Balakrishnan, N. and Cramer, E. (2014). *The Art of Progressive Censoring. Applications to Reliability and Quality*. Birkhäuser, New York.

Balakrishnan, N., Cramer, E., and Kundu, D. (2023). *Hybrid Censoring Know-How: Designs and Implementations*. Academic Press.

Balakrishnan, N. and Sandhu, R. A. (1995). A simple simulational algorithm for generating progressive type-ii censored samples. *Amer. Statist.*, 49:229–230.

Cramer, E. (2025). Structure of hybrid censoring schemes and its implications. *Metrika*, 88:393–417.

Cramer, J. I. and Cramer, E. (2026). Censopy – statistical computations forexponentially distributed lifetimes under hybrid censoring. submitted for publication.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

The mpmath development team (2026). *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.4.0)*. http://mpmath.org/.